RasPi

DESIGN
BUILD
CODE

6

Get hands-on with your Raspberry Pi

Master your PI CAMERA



65 PAGES OF PI

Plus

Program Minecraft Pi

Welcome



Happy new year, everyone! We're kicking off 2015 with a masterclass in one of the most popular add-ons for the Raspberry Pi – the camera module. Dave Jones, creator of the fantastic picamera library, shows us how to use his library to properly control your camera, even down to things like capturing motion and creating ring buffers to ensure you don't miss a trick. And as well as revisiting a couple of last year's projects, we thought we'd dig deep into the world of Minecraft Pi and show you how to manipulate it using scripts – especially now that this hack-focused edition of the game comes pre-installed on all new versions of Raspbian. Enjoy the issue – and have fun!

Gavin Thomas

Deputy Editor

Get inspired

Discover the RasPi community's best projects

Expert advice

Got a question? Get in touch and we'll give you a hand

Easy-to-follow guides

Learn to make and code gadgets with Raspberry Pi

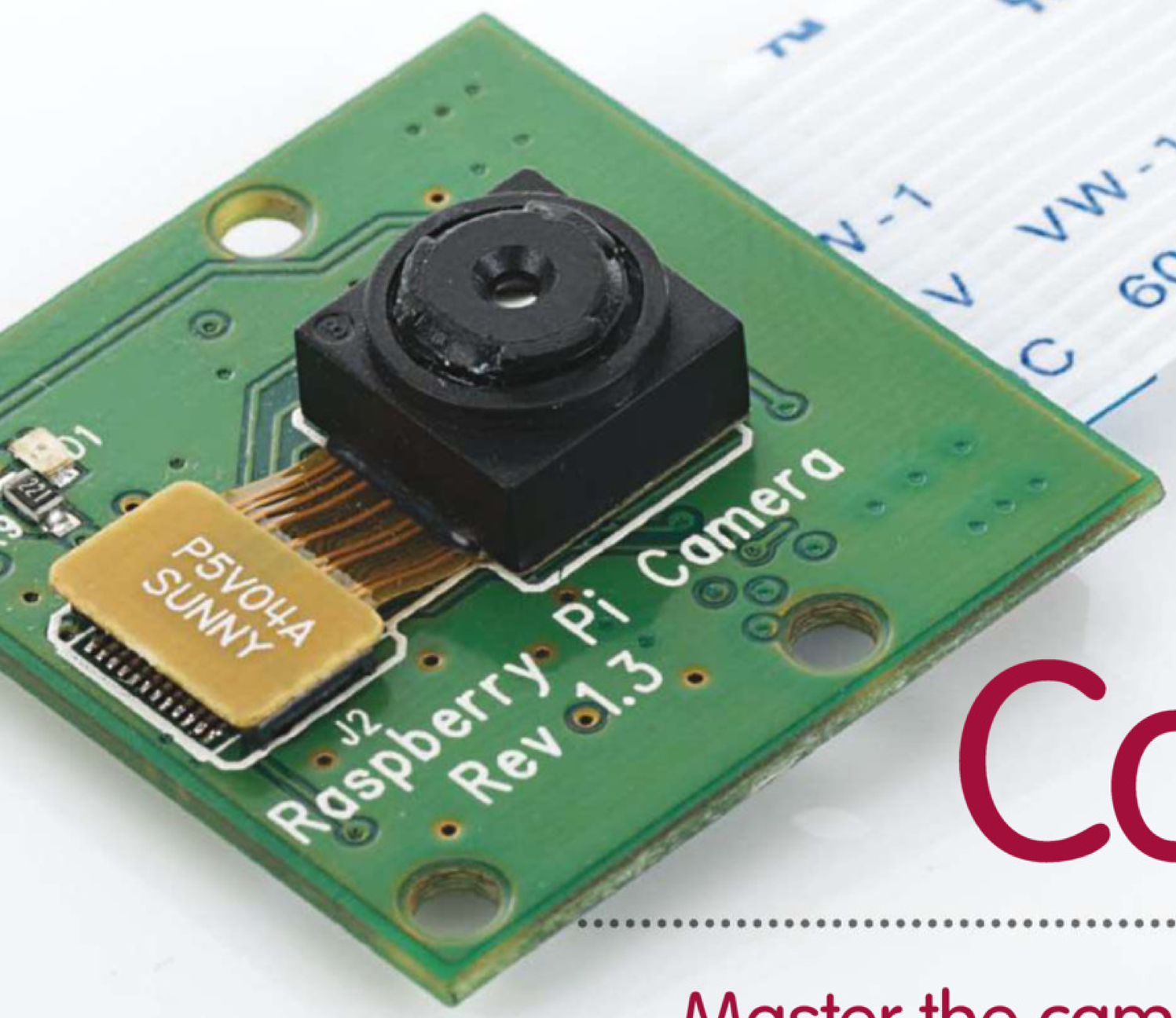


From the makers of

LinuxUser & Developer

Join the conversation at...

@linuxusermag | Linux User & Developer | @ RasPi@imagine-publishing.co.uk



Contents

Master the camera module
Motion detection and a time-travel camcorder

BrickPi Bookreader 2
Turn pages without lifting a finger

Upgrade your BigTrak
You can drive it – now fire its rocket launcher

What is the Raspberry Pi Compute Module?
Find out how this stick compares to the full board

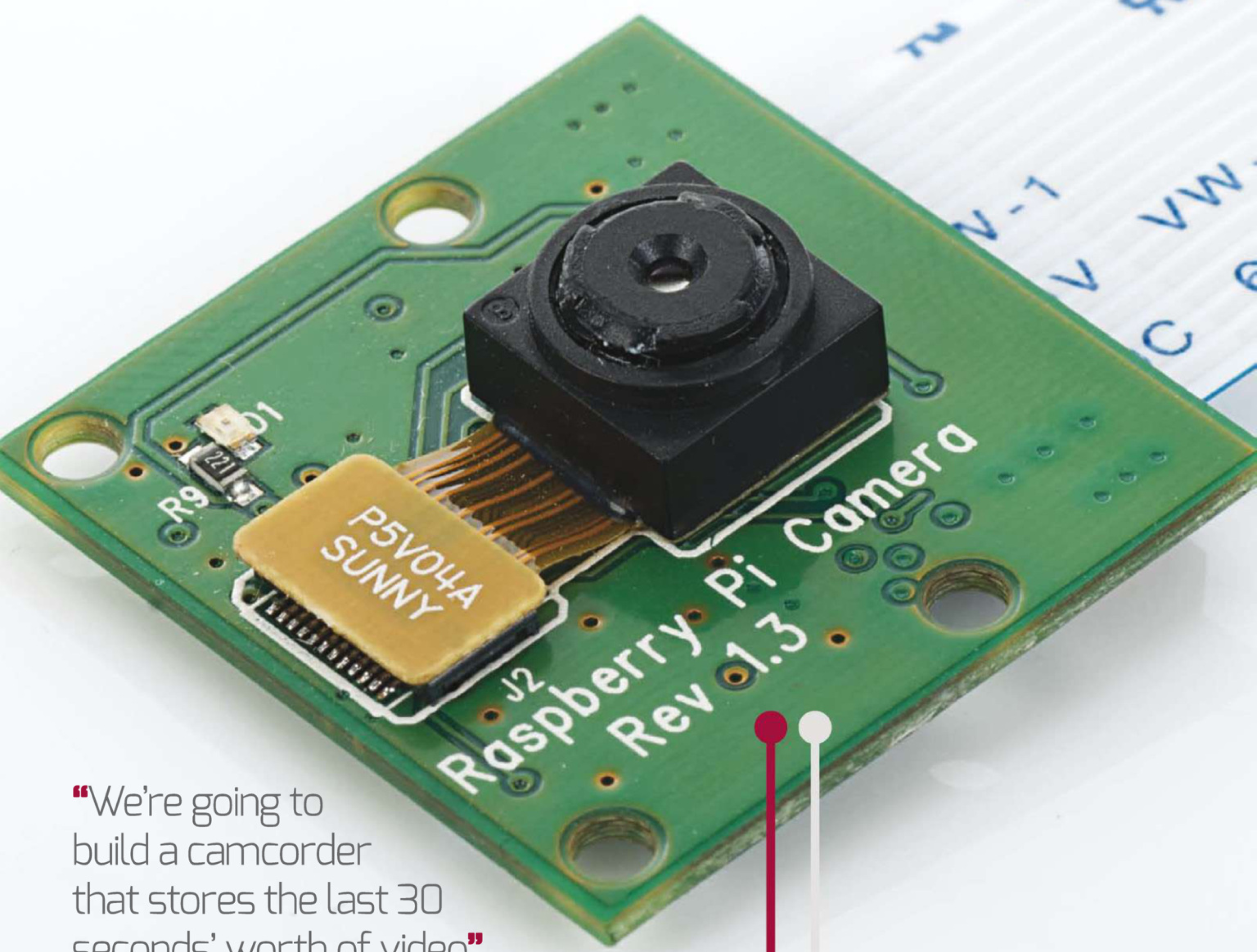
Write scripts for Minecraft Pi
Manipulate co-ordinates and auto-generate blocks

Looking at the world a different way
Read inputs from GPIO pins and trigger functions

Talking Pi
Your questions answered and your opinions shared

Master the camera module with the picamera library

Dave Jones walks us through how to set up and use his pure Python Raspberry Pi camera library

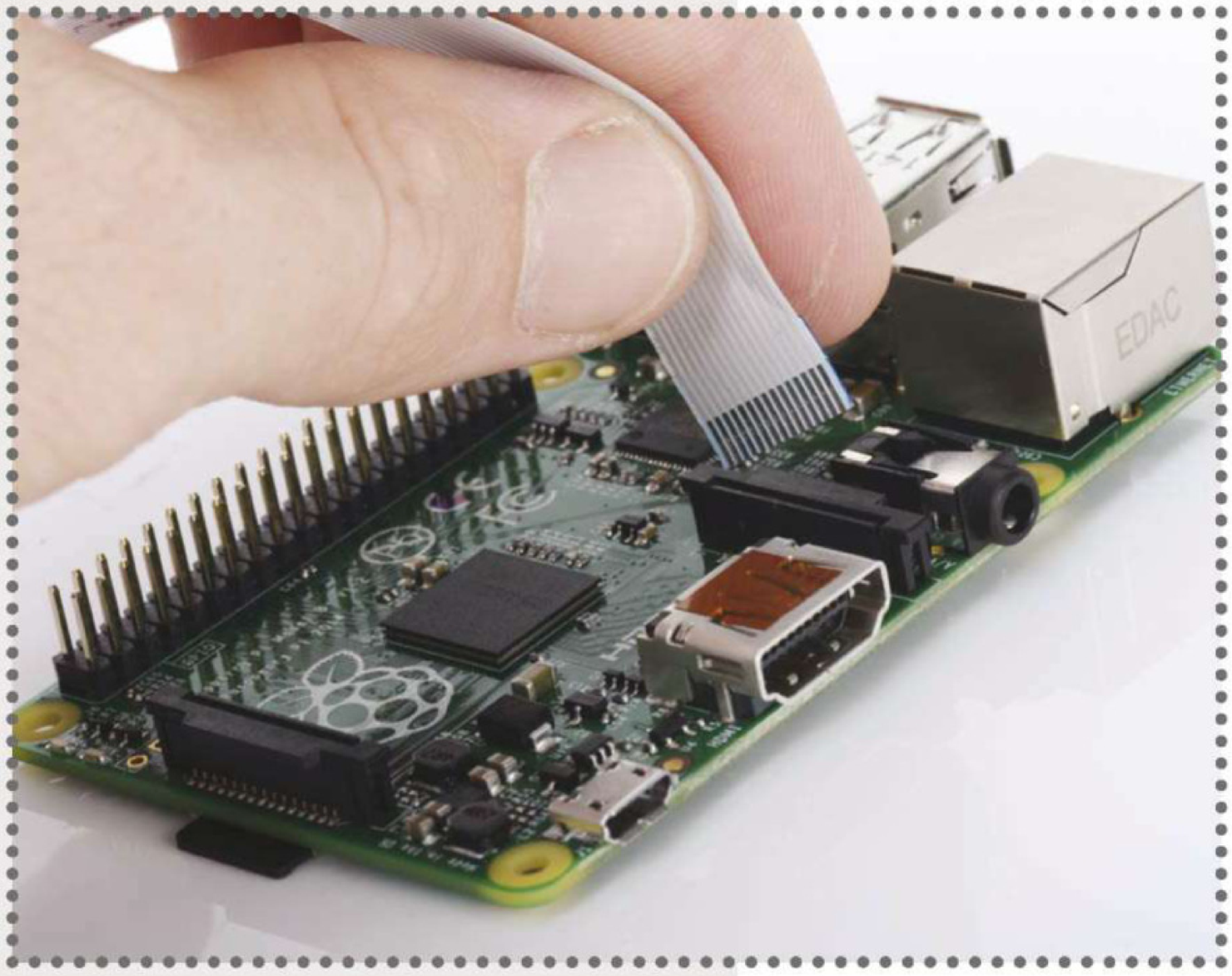


"We're going to build a camcorder that stores the last 30 seconds' worth of video"

The picamera library is a Python library permitting easy control of the Raspberry Pi's camera module. It was originally created to provide a cleaner method of controlling the camera than running the raspistill and raspivid command-line applications from within scripts. Since featuring on the Raspberry Pi homepage, the library has seen several thousand downloads and (naturally) a few bug reports – though as of release 1.2, these should all be fixed!

The library has already played a role in a number of fascinating projects, including time-lapse videos, motion-sensing security cameras, stop-motion animation, and even a handheld digital camera with Dropbox integration controlled by a touch screen. In this article we'll walk through installing and using the Raspberry Pi's camera module via picamera, including the fundamentals of two of the aforementioned projects: time-lapse videos and a motion-sensing security camera.

This article assumes you are using the Raspbian Linux distribution on your Pi, and that you are running the X Window environment (startx after logging in), although picamera works happily on any Pi-related Linux distro (Arch, Pidora, etc) and doesn't require a graphical environment, even for the camera's preview. It is also assumed you have successfully set up your camera module and tested it with one of the raspistill or raspivid applications.



> "Time-lapse videos, motion-sensing cameras, stop-motion animation, even a handheld digital camera"

Below If you need a hand setting up your camera module, check out the guide in issue 2 of **RasPi**

Getting Started

Start a terminal and run the following commands to install the picamera library for Python 2, along with some dependencies for the projects we'll be using later:

```
pi@raspberrypi ~ $ sudo apt-get update
pi@raspberrypi ~ $ sudo apt-get install
    python-picamera python-numpy python-rpi.
    gpio python-opencv ffmpeg
```

There's also a Python 3 package (python3-picamera), which happily coexists with the Python 2 package. All the code in this article will run under either version of Python (though not all third-party packages are available for Python 3 – OpenCV in particular).

Now that picamera is installed, you can try using it from an interactive Python session. Change to your Desktop directory, then enter the following:

```
pi@raspberrypi ~ $ cd ~/Desktop
pi@raspberrypi ~/Desktop $ python
Python 2.7.3 (default, Jan 13 2013, 11:20:46)
[GCC 4.6.3] on Linux2
Type “help”, “copyright”, “credits” or
    “license” for more information.
>>> import picamera
>>> camera = picamera.PiCamera()
>>> camera.capture(‘picture.jpg’)
>>> camera.close()
```

Finally, press Ctrl+D to exit the Python interpreter. You should now see a picture.jpg file on your desktop (which will have appeared after you ran the capture line in the session).

“There's also a Python 3 package (python3-picamera), which happily coexists with the Python 2 package”

Note that we explicitly closed the camera in the session above. The camera must always be closed before your script ends, but an easier way to accomplish this in most scripts is to use Python's with statement, like so:

```
import picamera

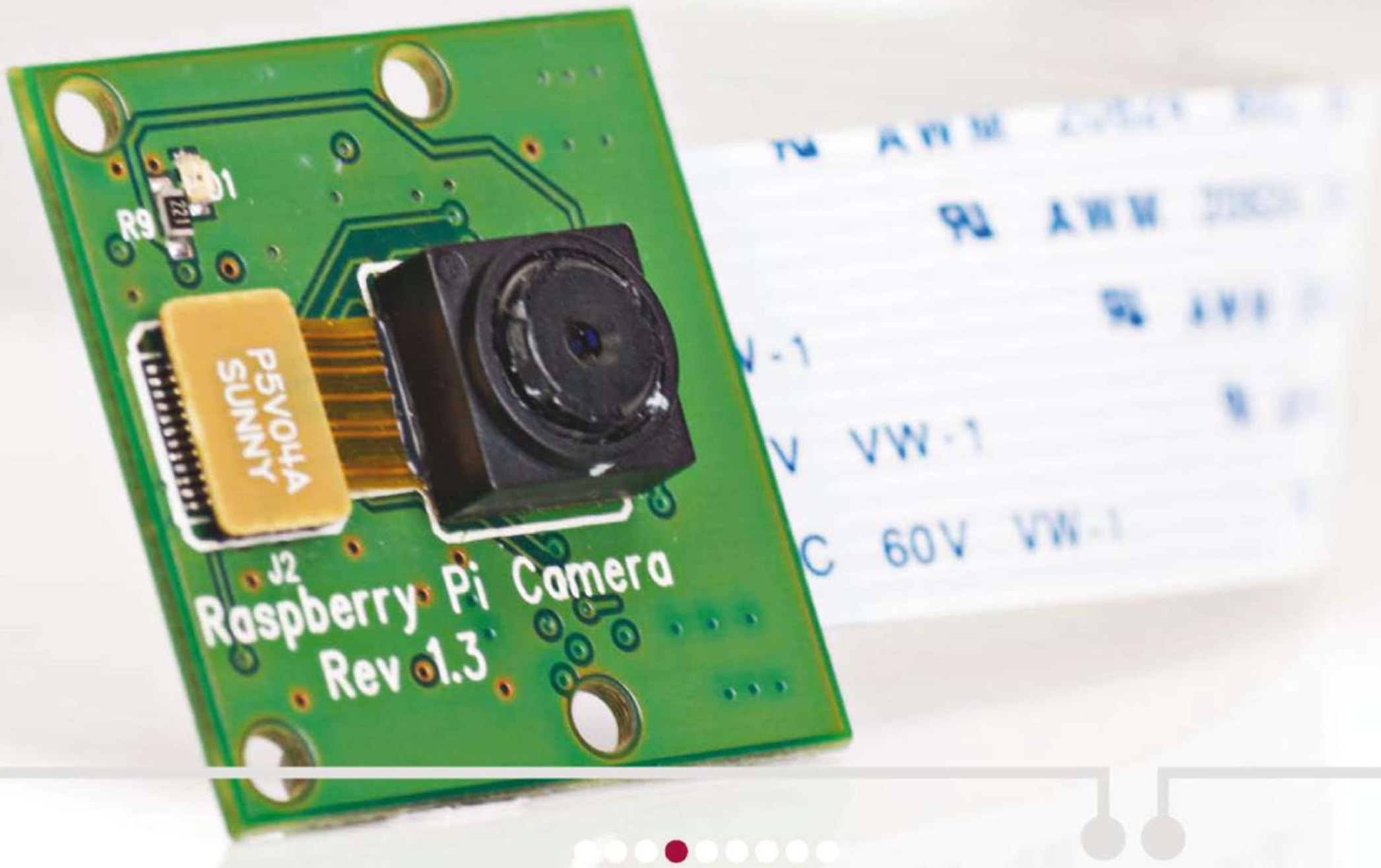
with picamera.PiCamera() as camera:
    camera.capture('picture.jpg')
```

When the with statement ends, the camera will automatically be closed.

Making time-lapse videos

One of the simplest but most rewarding projects you can undertake with the Pi's camera is creating a time-lapse video. There are many real-world processes that take place at such a slow speed that it's difficult for a human to appreciate what's going on. Fortunately, your Pi has considerably more patience! Pick an interesting subject

Below Try rigging up your Pi and camera module in a case and leaving it outside to shoot the night sky



to capture: the decay of a piece of fruit, the growth of a plant, the rise and fall of a nearby river, or perhaps the construction of a new building.

One crucial aspect of creating a time-lapse video is that the camera must remain reasonably still. A mount for the camera combined with a tripod can be invaluable here (see the 'Further reading' sidebar just a few pages forward from here for some suggested products). Try to locate the Raspberry Pi somewhere it is unlikely to be disturbed by anything, such as household pets or interested humans!

The script (**Fig 01** – at the end of this guide) will cause the Pi's camera to take a 1280x720 picture once every hour, for five days, storing the result in a series of incrementally named files. One picture per hour is a decent starting point for time-lapse videos, as a frame rate of 24 frames per second (common for video) will cause one day's worth of images (24 frames) to be compressed down to one second.

The constants at the top of the script can be manipulated to change the resolution of the resulting video, the capture rate of the script and the number of frames captured in total. Note that each time the script takes a picture, it initialises the camera and closes it again afterwards (remember that the with statement does this automatically). The camera significantly increases the power required by the Pi, so keeping it running all the time is not necessarily good, especially if you want to run your Pi on batteries for outdoor time-lapse sequences. You can run the script like so:

```
pi@raspberrypi ~ $ python timelapse.py
```

At the end of the capturing sequence, the script generates the time-lapse video by calling the FFmpeg utility. The Raspberry Pi is very slow at encoding the resulting video

Field of view

As you experiment with the camera, you may notice that it uses a larger field of view (FoV) for capturing images than when it is recording video or displaying a preview. You can force the preview to use the same FoV by setting the camera's resolution property to the maximum:

cam.resolution = (2592, 1944)

If you then wish to capture images at a lower resolution, use the resize parameter with the capture method:

cam.capture('foo.jpg', resize=(1024, 768))

The 'Camera Hardware' chapter in picamera's documentation explains the underlying reasons for the FoV discrepancy.

(typically it will take at least half an hour to do so), but given that the script has been running for five days at this point, that's probably not a huge burden!

The time-travel camcorder

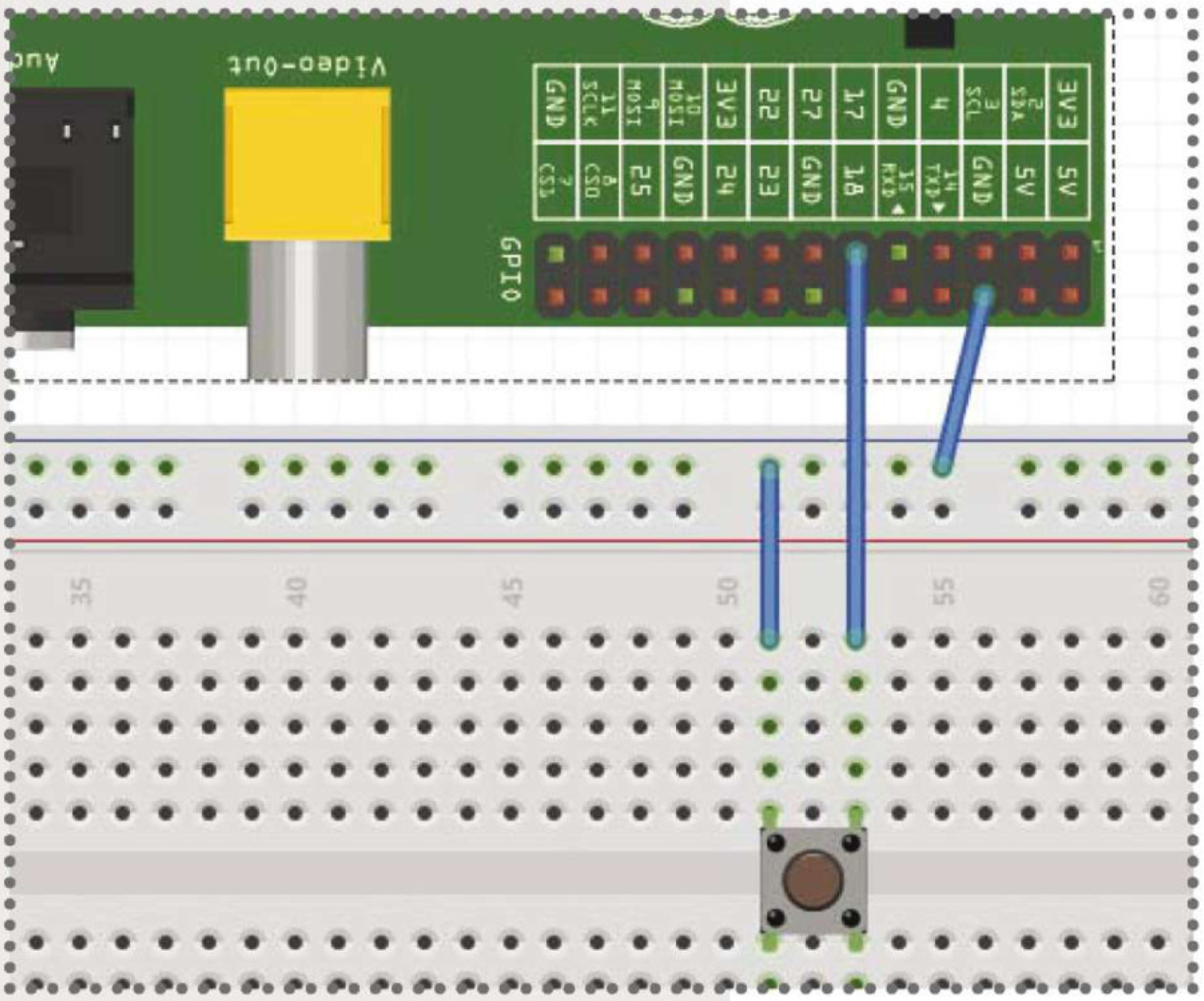
Disclaimer: no laws of physics were violated during the writing of this code!

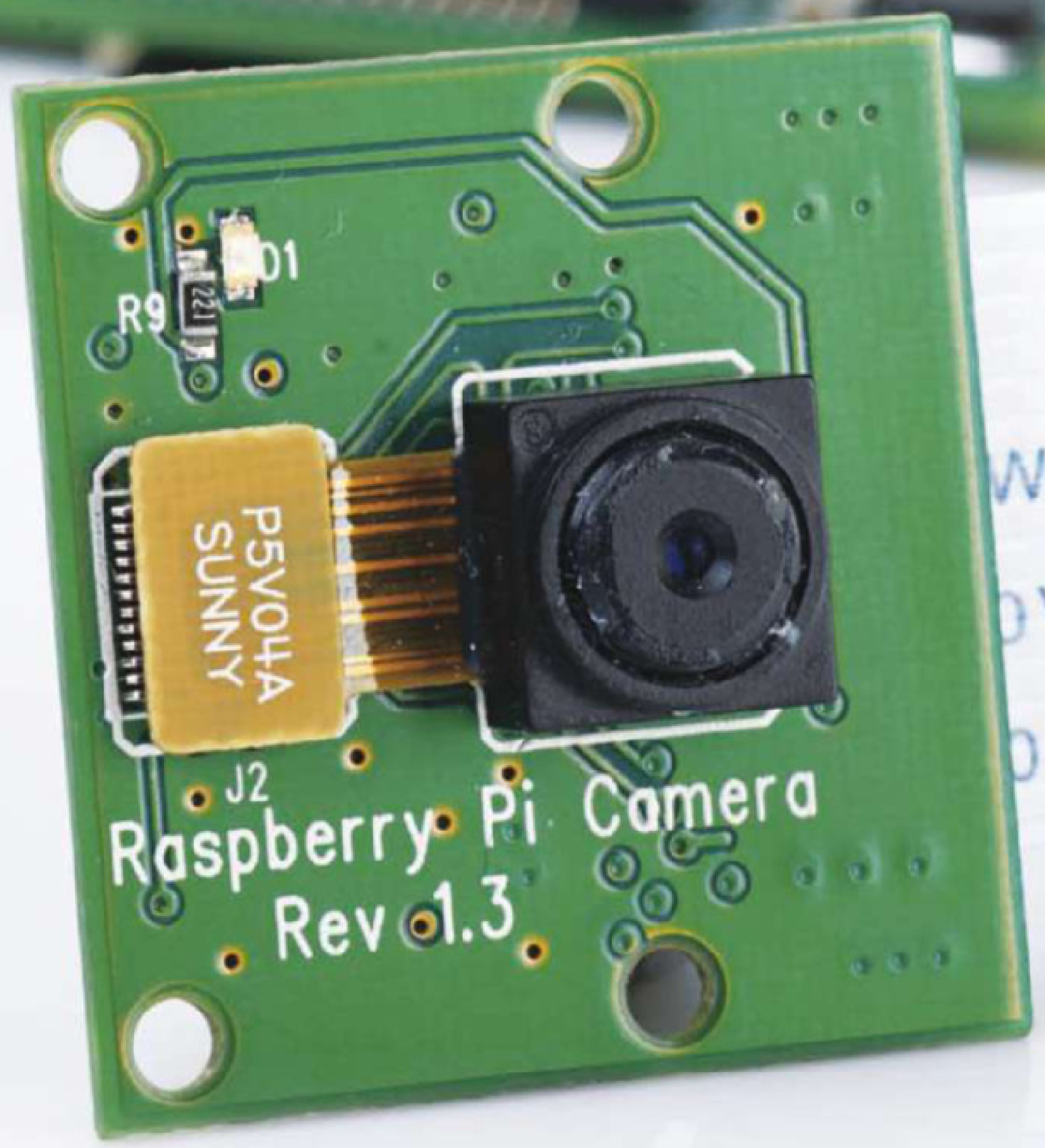
Next we're going to build a camcorder that, when activated, stores the last 30 seconds' worth of video (ie before the button was pressed). This may sound implausible, but what's going on under the hood is actually quite simple.

When the script begins, it immediately starts recording video to a special kind of stream called a 'ring buffer', which has enough space for about 30 seconds' worth of video. When a button is pressed, the script writes everything from the first header frame in the buffer to a file on the disk.

The ring buffer is provided by the picamera library in the form of the PiCameraCircularIO class. Under the hood it isn't actually a ring buffer. Rather, when the buffer is filled to capacity, it starts removing bits from the start of the buffer. The code listing (**Fig 02**) demonstrates what's going on. This script triggers the time-travel camcorder in response to a button connected to GPIO pin 17. The adjacent diagram shows a sample wiring setup using a standard breadboard.

Below Pimoroni sells some good buttons in three different sizes: **shop.pimoroni.com/products/tactile-switches**





Above Remember to keep your time-travel camcorder pointed at the action at all times –otherwise the ring buffer will be no use!

What use is a time-travel camcorder? There are obvious security applications: most motion-detection algorithms aren't perfect and by the time you've detected motion, you may have missed crucial details. Likewise for recording high-speed sporting moments, by the time you press the shutter button, the scene you were hoping to capture may have passed by. By keeping the last few seconds in memory, you can build a camera which helps you avoid missing the action!

Detecting motion

One of the more powerful algorithms you can implement with the Raspberry Pi's camera is motion detection. This capability can then be applied to any number of projects: the obvious security camera, recording video of wildlife, constructing a speed camera, or (when combined with peripherals connected to the GPIO port) controlling things in response to motion.

The classic method of motion detection is 'background subtraction'. First we capture an image and call it the

'background'. Then we capture another and subtract the pixel values of the first image from it. If nothing has changed, the resulting image will be entirely black. In practice, of course, things aren't quite so simple. The camera doesn't produce perfect images (there's a bit of 'noise'), so we introduce an erosion filter to reduce the noise, and a threshold to determine the cut-off point at which we decide motion has been detected.

Furthermore, we can't always use the same background image, otherwise a change to the scene (eg turning on a light in a room, or the movement of the sun for outdoor cameras) would result in us detecting motion all the time. We could use the image just before the one we're comparing, but this results in a motion detector which is quite easy to fool just by moving slowly (the tiny movements get wiped out by the erosion filter discussed above). Instead we use a moving average of the last few frames so that the background slowly adapts to changes.

In the script (**Fig 03**), to keep things simple we'll be working with greyscale images; specifically the Y channel of a raw YUV capture (see 'Further reading' for links on YUV image encoding). You can work with full-colour captures if you wish, but this significantly increases the amount of processing required and typically doesn't improve the sensitivity of the algorithm (think about a black and white movie versus a colour one – do you notice motion more easily in colour?). You can run the script like so:

```
pi@raspberrypi ~ $ python motion.py
```

When the script is running, try waving your hand in front of the camera. With any luck you should see the terminal print a message in response. Try to fool the motion-detection algorithm; by moving very slowly, you may be able to do

Further reading

The GitHub home for picamera; you can 'watch' the project for updates from here:

github.com/waveform80/picamera

This is picamera's documentation, including numerous recipes covering a range of skill levels:

picamera.readthedocs.org/

Installation instructions for the Pi's camera module:

www.raspberrypi.org/camera

A camera mount (eg for a tripod) is extremely useful for a variety of camera projects:

www.modmypi.com/raspberry-pi-camera/pibow-raspberry-pi-camera-mount

so. Have a play around with the values that control the sensitivity of the algorithm – in particular background_delta, erosion and threshold.

As a rule of thumb, resolution has little to do with motion detection. You can use quite low resolutions (such as 160x90, as in the script above) and still achieve reasonable results, assuming that the thing you want to detect the motion of (eg a human, or an animal) still takes up a significant portion of the camera's field of view.

However, if you want to capture images or record video in response to detecting motion (a typical requirement of a security camera), you probably want a higher resolution than 160x90. This is why the script above sets the camera to a high resolution (1280x720), then uses the resize option of the capture method to downsize the images required for the motion-detection algorithm.

One final thing to note is that this script is using video-port-based captures. While this provides a decent capture rate (up to the frame rate of video captures), it does result in a reduced field of view because, as far as the camera is concerned, we are capturing video (see the 'Field of view' sidebar just four swipes back for hints on mitigating this).

There are a couple of recipes in the picamera documentation that call for a motion-detection routine, and which utilise the circular buffer introduced above. Try combining the code listed on the next few pages with those recipes to produce a security camera that only records footage when it detects motion.

"As a rule of thumb, resolution has little to do with motion detection. You can use quite low resolutions such as 160x90"

Further reading

Some Pi cases have integral camera mounts, and even enhanced lenses:

www.modmypi.com/raspberry-pi-camera/nwazet-pi-camera-box-bundle

The GorillaPod is an excellent portable tripod:

joby.com/gorillapod

YUV image encoding is used by the Pi's camera internally, and is the most efficient form of raw capture:

en.wikipedia.org/wiki/YUV

The Code FIG 01 TIMELAPSE.PY LISTING

```
from __future__ import division

import time
import subprocess
import picamera

VIDEO_RESOLUTION = (1280, 720)
VIDEO_DAYS = 5
FRAMES_PER_HOUR = 1
FRAMES = FRAMES_PER_HOUR * 24 * VIDEO_DAYS

def capture_frame(frame):
    with picamera.PiCamera() as cam:
        cam.resolution = VIDEO_RESOLUTION
        # Give the camera a couple of seconds to settle
        # before capturing. This is generally a good
        # idea immediate after initializing the camera
        time.sleep(2)
        cam.capture('frame%03d.jpg' % frame)

# Capture the images
for frame in range(FRAMES):
    # Note the time before the capture
    start = time.time()
    capture_frame(frame)
    # Wait for the next capture. Note that we take into
    # account the length of time it took to capture the
    # image when calculating the delay
    time.sleep(
```

"The script will cause the Pi's camera to take a 1280x720 picture once every hour, for five days, storing the result in a series of incrementally named files"

The Code FIG 01 TIMELAPSE.PY LISTING

```
        int(60 * 60 / FRAMES_PER_HOUR) - (time.time() - start)
        )

# Generate the video
subprocess.call([
    'ffmpeg', '-y',
    '-f', 'image2',              # Use images as the source
    '-i', 'frame%03d.jpg',       # The input filename pattern
    '-r', '24',                  # The frame-rate for the output
    '-vcodec', 'libx264',        # Use x264 to encode the output
    '-profile', 'high',          # Permit high-profile H.264 features
    '-preset', 'slow',           # Use a good quality preset
    'timelapse.mp4',             # The output filename
    ])
```

The Code FIG 02 TIMETRAVEL.PY LISTING

```
#!/usr/bin/env python

import io
import picamera
import RPi.GPIO as GPIO
import datetime as dt

BUTTON_PIN = 17

def button_callback(channel):
    assert channel == BUTTON_PIN
    print('Button pressed')
    filename = dt.datetime.now().strftime('%Y%m%d-%H%M%S.h264')
    # Lock the circular buffer, and open the output stream as
    # a binary file
    with stream.lock, io.open(filename, 'wb') as output:
        # Find the first header frame in the circular buffer
        for frame in stream.frames:
            if frame.header:
                stream.seek(frame.position)
                break
        # Efficiently copy from this position to the end of
        # the circular buffer to the output file
        while True:
            data = stream.read1()
            if not data:
                break
            output.write(data)
        print('Wrote video to %s' % filename)
```

"It starts recording video to a special kind of stream called a 'ring buffer', which has enough space for about 30 seconds' worth of video"

The Code FIG 02 TIMETRAVEL.PY LISTING

```
        # Wipe the content of the circular buffer so we don't
        # repeat any frames if the button is pressed again
        # quickly
        stream.seek(0)
        stream.truncate()

# Set up the button as an input
GPIO.setmode(GPIO.BCM)
GPIO.setup(BUTTON_PIN, GPIO.IN, GPIO.PUD_UP)

# Start the camera recording to an in-memory ring buffer with
# space for 30 seconds worth of video at 720p
with picamera.PiCamera() as cam:
    cam.resolution = (1280, 720)
    stream = picamera.PiCameraCircularIO(cam, seconds=30)
    cam.start_recording(stream, format='h264')
    GPIO.add_event_detect(BUTTON_PIN, GPIO.RISING,
            callback=button_callback, bouncetime=200)
    try:
        while True:
            cam.wait_recording(1)
    finally:
        GPIO.remove_event_detect(BUTTON_PIN)
        cam.stop_recording()
```

The Code FIG 03 MOTION.PY LISTING

```
#!/usr/bin/env python

from __future__ import division

import io
import time
import picamera
import cv2
import numpy as np

class MotionDetector(object):
    def __init__(self, camera, resolution=(160, 90), threshold=20,
                 erosion=10, background_delta=0.3):
        self.camera = camera
        self.resolution = resolution
        self.raw_resolution = (
            (resolution[0] + 31) // 32 * 32,
            (resolution[1] + 15) // 16 * 16,
            )
        self.raw_bytes = self.raw_resolution[0] * self.raw_resolution[1]
        self.threshold = threshold
        self.erosion = erosion
        self.background_delta = background_delta
        self.background = None

    def _get_erosion(self):
        return (self.erosion_filter.shape[0] - 1) // 2
    def _set_erosion(self, value):
        self.erosion_filter = cv2.getStructuringElement(
            cv2.MORPH_RECT, (value * 2 + 1, value * 2 + 1))
    erosion = property(_get_erosion, _set_erosion)
```

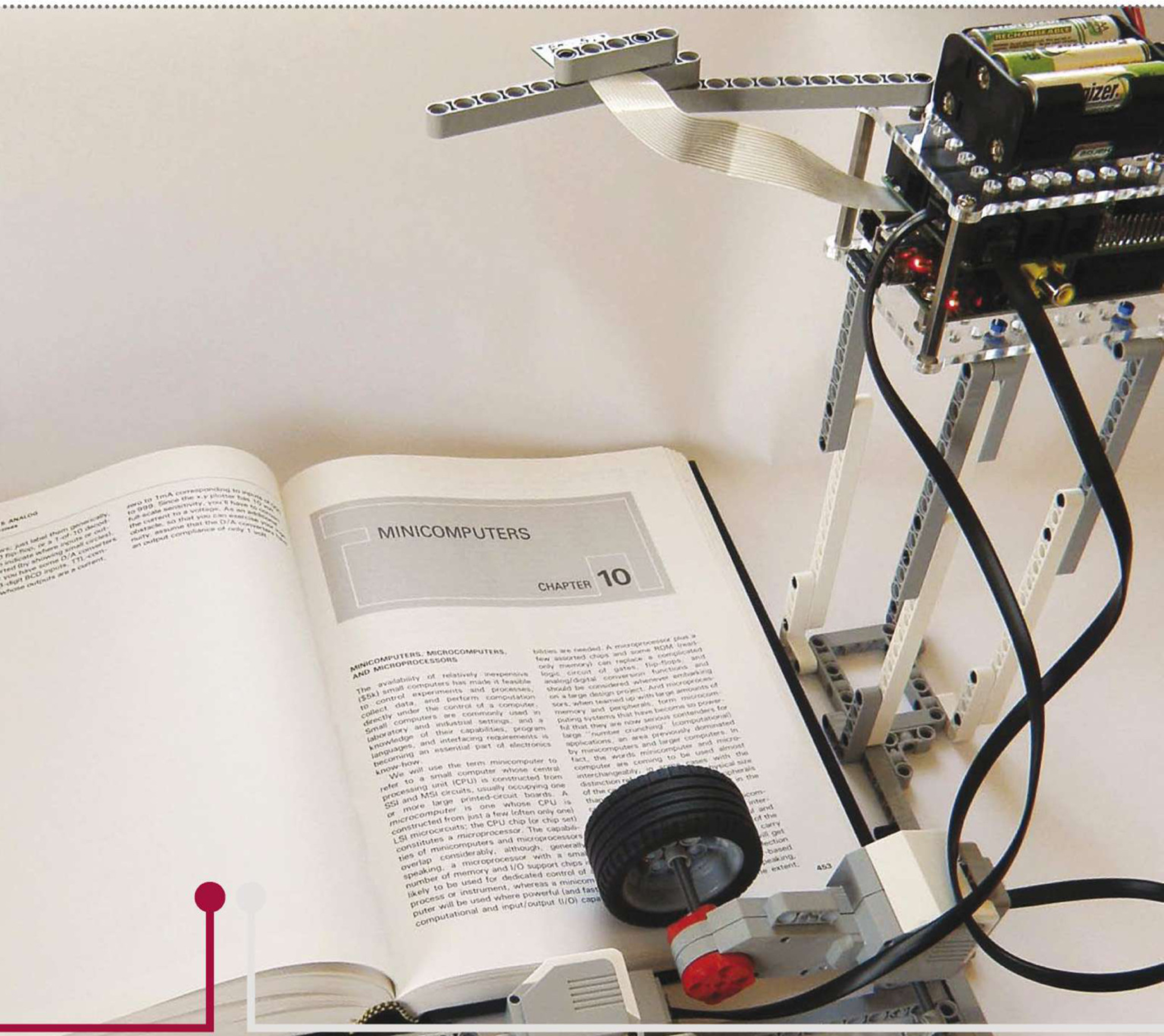
"We use a moving average of the last few frames so that the background slowly adapts to changes"

The Code FIG 03 MOTION.PY LISTING

```
    def poll(self):
        stream = io.BytesIO()
        self.camera.capture(
            stream, format='yuv', resize=self.resolution, use_video_port=True)
        data = stream.getvalue()[:self.raw_bytes]
        image = np.fromstring(data, dtype=np.uint8).reshape(
                (self.raw_resolution[1], self.raw_resolution[0])
                )
        cv2.erode(image, self.erosion_filter)
        image = image.astype(np.float)
        if self.background is None:
            self.background = image
            return False
        else:
            diff = cv2.absdiff(image, self.background)
            diff = diff.astype(np.uint8)
            diff = cv2.threshold(
                    diff, self.threshold, 255, cv2.THRESH_BINARY)[1]
            result = diff.any()
            cv2.accumulateWeighted(
                    image, self.background, self.background_delta)
            return result
with picamera.PiCamera() as cam:
    cam.resolution = (1280, 720)
    # Let the camera warm up before we start
    time.sleep(2)
    detector = MotionDetector(cam)
    while True:
        if detector.poll():
            print("I see you!")
```

BrickPi Bookreader 2

A robotic, mechanical reader of printed books that melds together the Raspberry Pi and Lego Mindstorms



Where did the idea of the Bookreader 2 come from?

When the Raspberry Pi camera module was originally released we thought it would be great to show that you can use the camera with some of the Lego robots and that there's a whole lot of awesome stuff you can do with it. We put something together that was just an arm to turn the Kindle and the camera to read it aloud; we thought that would be really interesting but it got the wrong type of attention. We got a lot of comments saying, "Well there's already stuff that can do that". People missed the point that we were trying to show a tour de force with a Raspberry Pi. So we doubled down and decided we just had to put together a mechanical version that would turn pages – just so we could make our point [laughs]. So the first Bookreader did the Kindle and we read a book off of it out loud and people were like, "Well that's cool, but there's already software to do that".



John Cole is a chemical engineer by trade but decided to try his hand at robotics. In 2010 he started Dexter Industries to create and sell robotic products

It uses your BrickPi add-on for the Raspberry Pi – what exactly is that?

The BrickPi is a platform for connecting the Raspberry Pi and Lego Mindstorms. So it connects the old NXT, last year's generation, along with this year's generation (the EV3) to the Raspberry Pi – so you suddenly have a very powerful and very open Linux system robot which can be configured in a trillion different ways and can build to your heart's content.

Why create the BrickPi for the Raspberry Pi?

The Raspberry Pi is great. I guess there are other Linux systems out there, including TI's BeagleBone, but Raspberry Pi has two huge things going for it and the

first one is it's the most widely distributed and most easily accessible system. It's sort of default when people want to talk about SoC systems running Linux. The other thing is the Raspberry Pi Foundation and the Raspberry Pi in general has cultivated a huge educational following. That's been their stated wheel-house – that they want to do education – and that's what they've done and that's sort of how the community has grown up around it. We like to think of ourselves really as an education company and it's the best way to learn robotics. Raspberry Pi is behind that and we wanted to be part of that, which is one of the reasons why we chose it. It's got a very open and active community as well, which ties into a lot of people using it – but the fact there are so many people using it and contributing back to it makes developing new stuff and mashing up other hardware with the Raspberry Pi quite easy because the community had already laid the foundation for us to do that.

How long did it take to develop the Bookreader 2 after the first Bookreader?

Oh, it was a couple of months. We put the Bookreader out just before Christmas I believe, and then the holidays came so we were busy with taking care of logistics stuff and having a holiday. So after Christmas we found a couple of extra hours to do the programming and redo the hardware; it looks really cool but it was a really simple mashup. The optical character recognition software is already out there and there are tons of tutorials on how to get that to read text off of a page and read that sort of stuff, so all we did was mash it up and make the Lego model and we were in business.

If you like

The BrickPi is one of the core components of the Bookreader and a great way to learn some basic robots using this and LEGO Mindstorms

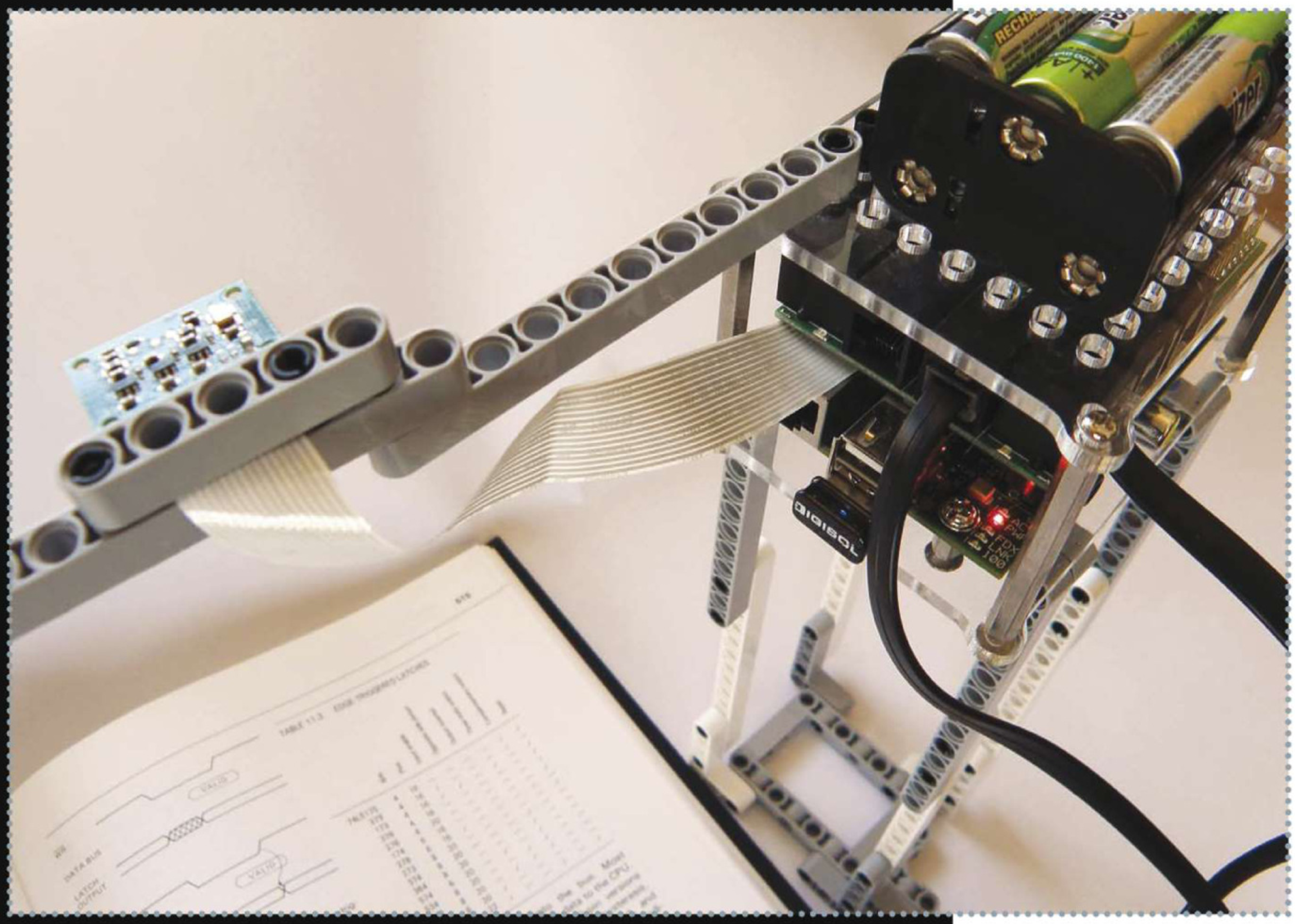
Further reading

To learn more about the Bookreader 2 and BrickPi visit: **dexterindustries.com**

Did you have problems with the page-turning functionality via the wheel?

No. We actually saw there were a couple of folks that had done it with previous Lego systems and they'd done it a little bit differently. We wanted to use the EV3 system because the BrickPi is associated with Lego NXT, yet it also works with the EV3, so we just had to make some modifications to it and get it to work. The hard part is the timing in something sort of physical, and I think if we were to redo this again we would have the camera control the motors and have some detection of whether the page is fully turned or not. There are a thousand directions we could take it right now.

Below Looking somewhat like a section in a Rube Goldberg machine, the Bookreader is a fully functional robot

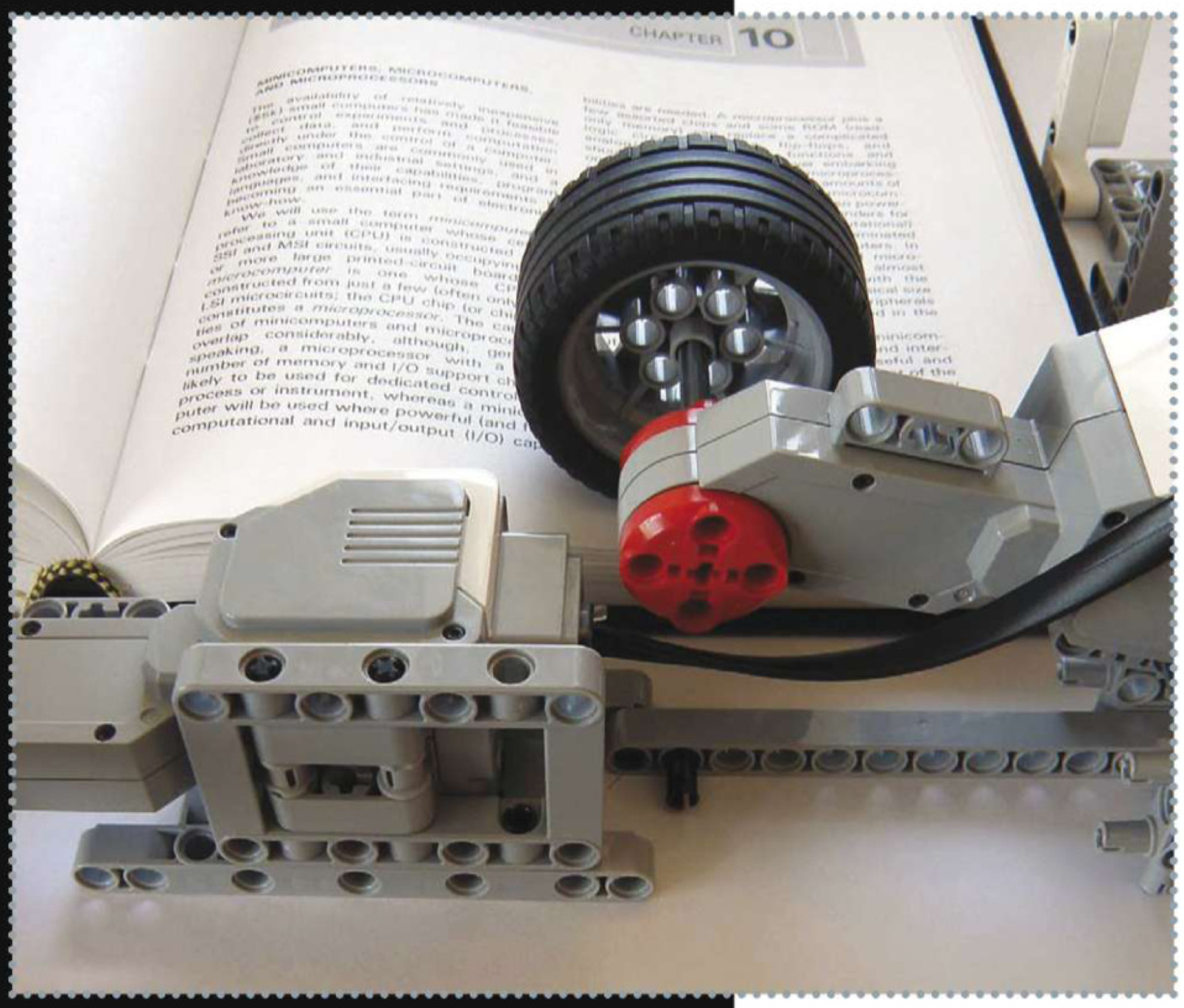


What plans do you have for future Raspberry Pi projects and BrickPi?

When we were using the NXT while developing the BrickPi one of the limitations was just to get started with the BrickPi you needed a Lego Mindstorms kit. So it was sort of one our project developments to come up with an easy platform that somebody could use with little to no wiring and certainly no soldering. The camera package that came out last year for the Raspberry Pi was sort of the inspiration because people found ways to hack it – but there wasn't really an out-of-the-box easy solution where you could just put it together. My dad, who is an old-time computer programmer, wouldn't touch a soldering iron if you paid him.

There is that barrier for entry where maybe there's some intimidation and it's hard to get started. The GoPiGo is a package you can pull out of the box and use nothing but a screwdriver to get started… it opens up robotics to a new set of people, which is what we've tried to do – and I think we've done that.

"The GoPiGo is a package you can pull out of the box and use nothing but a screwdriver to get started"



Below The wheel turns the pages individually, although it does it via precise timing and not sensing for the moment

Upgrade your Pi-powered Bigtrak

Get your Bigtrak ready for action by arming it with the Bigtrak Rocket Launcher!



When the Bigtrak was re-released back in 2010 it was accompanied with two accessories, the Can Caddy and the Rocket Launcher. The Can Caddy is a couple of pieces of plastic that mounts onto the back of the Bigtrak and allows it to carry around a drinks can. But it's not very exciting and the top-mounted Raspberry Pi gets in the way of fitting it.

The Rocket Launcher, however, is a much more interesting item, consisting of four foam-tipped rockets that you can program the Bigtrak to fire as it trundles around. Of course, as you have to pre-program the Bigtrak's route, it gives your target ample opportunity to get out of the way.

To rectify this flaw we're going to take the Raspberry Pi-controlled Bigtrak that we put together in the previous issue and update it to fire rockets whenever you press 'X' on the controller.

01 Trigger the rockets

There are two ways of triggering the rocket to fire. The first, used by the Bigtrak Junior, is achieved by running a 4.5V current through the 3.5mm jack plug. The second, used by the Bigtrak, is to send an infrared signal to the Rocket Launcher in a similar way to how a TV remote works.

02 Power switch and IR LED

The Bigtrak's IR LED sits together with the power switch on a small PCB with a multicoloured ribbon cable running from it. Since we are only interested in the IR LED, we need to determine which cables connect to it. This can be done by following the traces running from the IR LED to the cables, or by using a multimeter.

THE PROJECT ESSENTIALS

Pi-powered Bigtrak
See issue 05

Bigtrak rocket launcher
bit.ly/1nRBjFo

USB Battery pack
amzn.to/1h2PBiI

NPN transistor

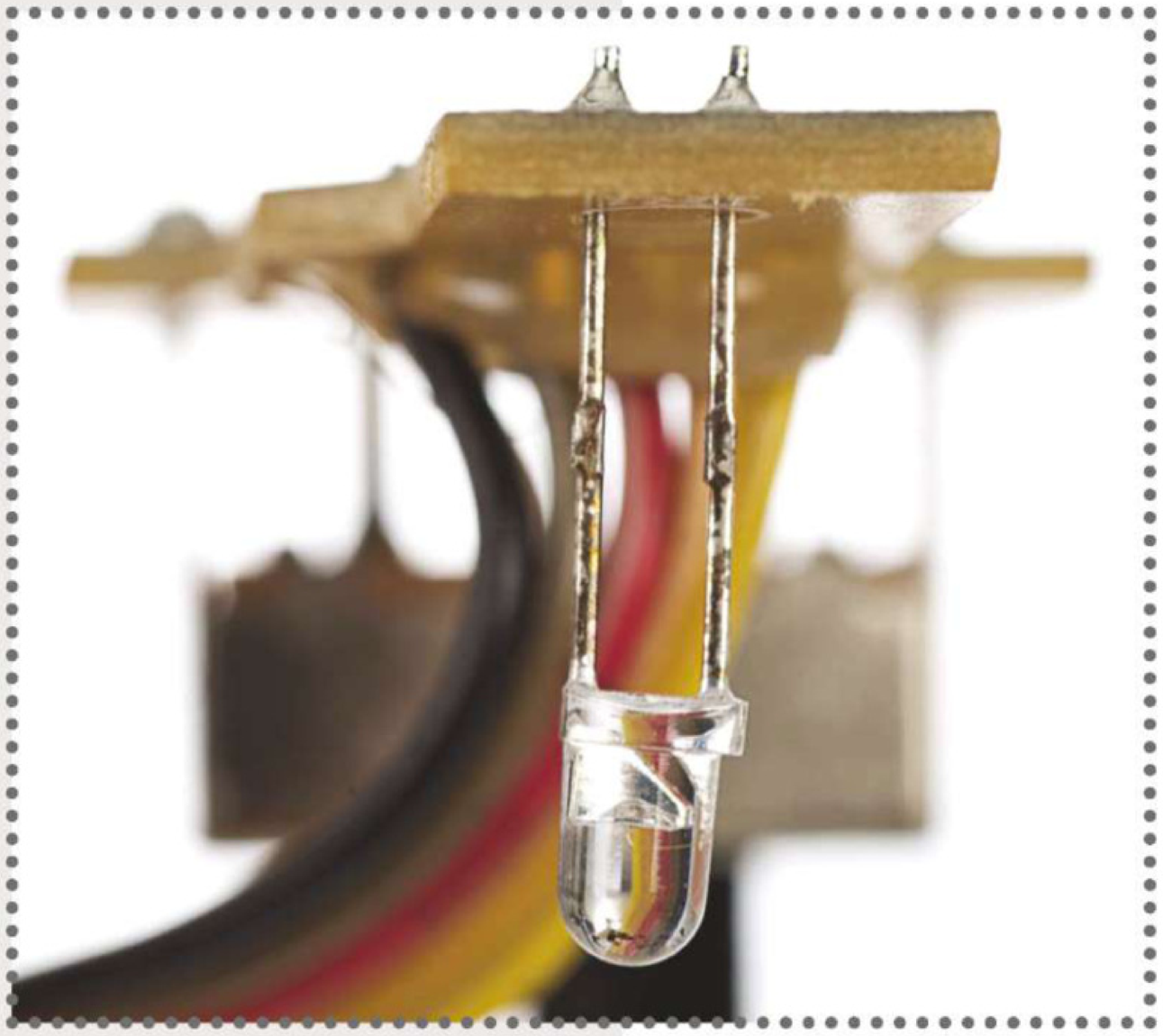
47Ω and 470Ω resistors

Driving a Bigtrak with your Pi

Last issue, we showed you how to connect up your Raspberry Pi to the Bigtrak in order to drive it using a PS3 controller. This tutorial follows on from that point, so you'll need to go back and give the other guide a read first so that you've got the basic driving controls sorted.

03 Determine the polarity of the LED

Before we attach the IR LED to the chassis we need to determine which of the wires connect to the anode (positive) leg and which connects to the cathode (negative) leg of the LED. Looking closely at the IR LED, you can see that one side is flatter than the other: this is the side the cathode pin is on.



Above The base is flat on the negative side, round on the positive

04 Wire up the IR LED

Now that we know which two wires to use, they can be soldered onto a connector. We've used a module connector for ease of connecting to the breadboard, along with some heat shrink wire wrap to protect the soldered joints, which will reduce the chance of an accidental short circuit.

05 Re-attach the power switch

Turn the lid of the Bigtrak upside down and locate the mounting point for the power switch PCB. Carefully put the PCB into place and then screw it down to hold it securely.

06 Transistors and resistors

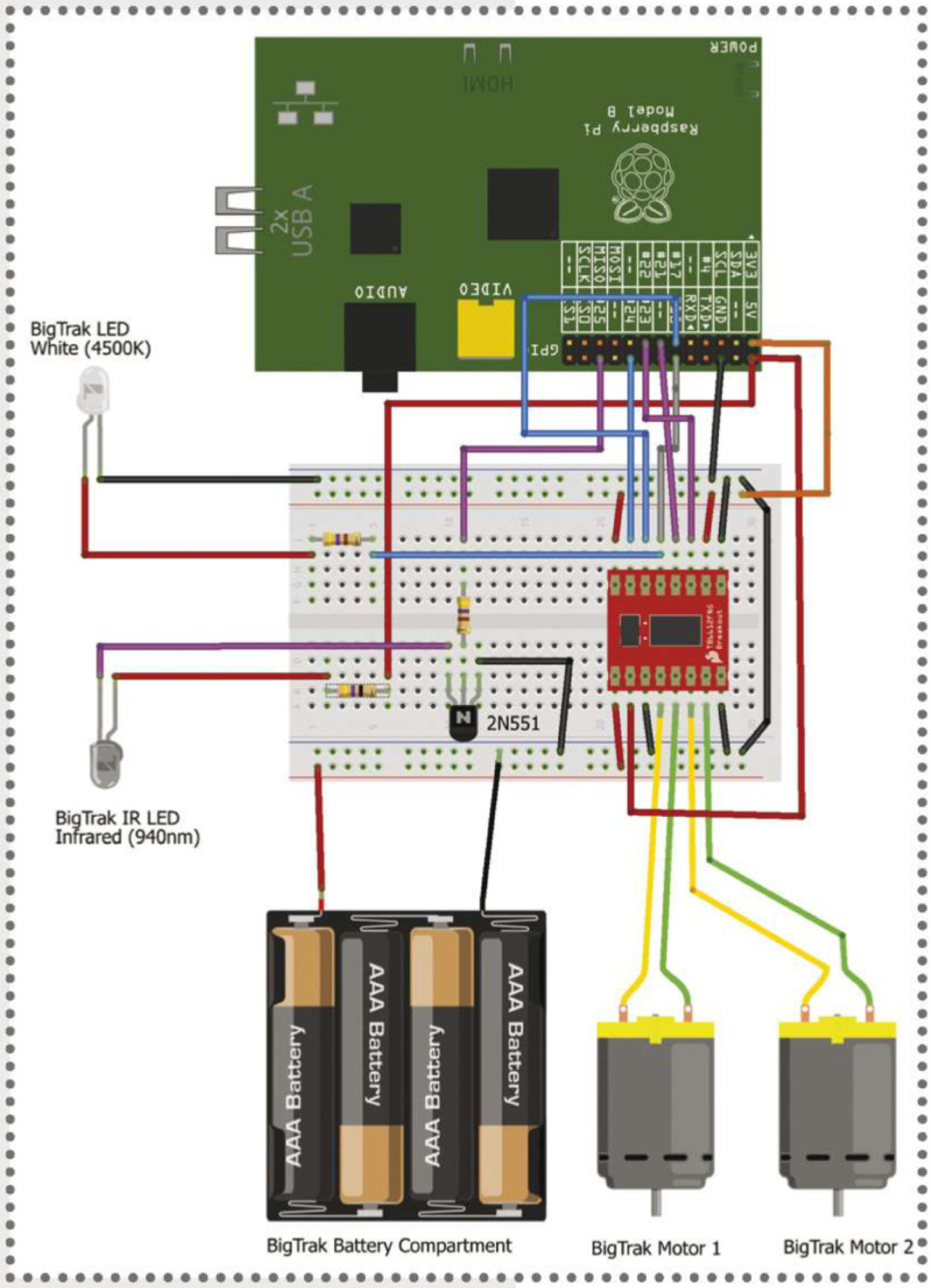
The IR LED is going to be powered from the Raspberry Pi's 5V line, so to protect the GPIO pins from this higher voltage we'll be using a 2N5551 transistor to amplify the output from the GPIO pin, as well as several resistors to prevent too much current being drawn.

Power switch and IR LED

When working on projects, we keep any leftover components in a 'bits' box. If you modified a Bigtrak last issue and didn't keep the power switch, or if you just want to fire the rockets without the Bigtrak, you can get a replacement IR LED that can be held in place with some electrical tape or sticky tack.

07 Wire up the components

Transistors have three legs called base, emitter and collector. We connect the emitter to the ground, the base (via a 470Ω resistor) to GPIO pin 25 and the collector connects to the cathode leg of the IR LED. To complete the circuit, the anode leg of the LED needs to be connected (via a 47Ω resistor) to the 5V pin on the Raspberry Pi.



Above Look familiar? We're just expanding on last issue's circuit

08 Install the software

LIRC (Linux Infrared Remote Control) is a collection of utilities that allows the Raspberry Pi to send and decode IR signals. We'll be using the 'irsend' utility to launch the rockets. To install lirc, run:

```
sudo apt-get install lirc
```

09 Enable the LIRC

To enable LIRC and to tell it which GPIO pin to use as the transmitter run, use:

```
sudo nano /etc/modules
```

...and add the following lines, changing the value of gpio_out_pin to the one that the IR receiver is connected to and setting the gpio_in_pin to an

unused pin. This version of the lirc_rpi module will require both in and out to be set.

```
lirc_dev
lirc_rpi gpio_in_pin=4 gpio_out_pin=25
```

10 Configure the LIRC

LIRC supports multiple ways of controlling IR, so it needs to be configured to use the Raspberry Pi GPIO pins. To do that, enter the following command:

```
sudo nano /etc/lirc/hardware.conf
```

…and manually change the DRIVER, DEVICE and MODULES lines to:

```
DRIVER="default"
DEVICE="/dev/lirc0"
MODULES="lirc_rpi"
```

11 Teach LIRC how to fire the rocket

To fire the rocket we need to send the correct set of pulses through the IR transmitter to trigger it. To learn these pulses, LIRC provides the 'irrecord' utility which, when combined with an IR receiver, allows us to record the IR pulses sent by an unmodified Bigtrak when it triggers its 'fire' instruction. This will generate a configuration file that, after a few minor tweaks, looks like the one in the lircd.conf code listing at the end of this tutorial. You can manually enter these values by running…

```
sudo nano /etc/lirc/lircd.conf
```

…and exactly typing in the text on the last page of the code listing that starts at the end of these steps. Don't worry – all this precise typing is great practice for the command line!

Transistors & resistors

If you are using a different transistor then the value of the resistors may change. One guide for calculating them can be found at **bit.ly/1qqtaqg** and the datasheets for these components can usually be found with a quick search.

12 Test the LIRC setup

To enable the changes, we need to reboot the Raspberry Pi by entering…

```
sudo reboot
```

…and, after rebooting, we can check that LIRC has correctly loaded by typing:

```
dmesg | grep lirc
```

13 Test fire the rockets

To check the hardware and software setup, we can fire the rockets manually by entering:

```
irsend SEND_ONCE bigbrak fire
```

If all has gone well then one rocket will launch. By running the command again you can fire off the rest of the rockets.

14 Add extra setup code to the Python script

To trigger the Rocket Launcher from the PS3 controller we need to make some simple changes to the Bigtrak.py Python script that we used in the last issue. To run the 'irsend' command from Python we will be using the 'subprocess' module, and we'll be using the 'X' button on the controller to trigger it. So towards the top of the Bigtrak.py file we need to add:

```
import subprocess
PS3_CROSS = 14
```

> "To trigger the Rocket Launcher from the PS3 controller we need to make some simple changes to Bigtrak.py"



15 Response to the button press

We want the rocket to respond to our input and fire when the 'X' button is pressed, so after the existing code that checks if the analog sticks have been moved we just need to add the following lines:

```
elif event.type == pygame.JOYBUTTONDOWN:
  if event.button == PS3_CROSS:
    subprocess.call(["irsend","SEND_ONCE","Bigtrak","fire"])
```

16 Run the script

With the code complete we can now run it with:

```
sudo python BigtrakWithRockets.py
```

If all has gone well, you can now drive the Bigtrak around and launch rockets on demand!

17 Next steps

With our rocket-toting Bigtrak, we're now a little closer to our plans for world domination (although we might be needing some bigger rockets for that). We've used one of the official accessories here, but there are plenty of unofficial accessories we could add. Ultrasonic sensors, a robot arm or the Raspberry Pi camera, perhaps?

"There are plenty of unofficial accessories we could add. Ultrasonic sensors, a robot arm or the Raspberry Pi camera, perhaps?"

Having trouble?

If the rocket doesn't fire then try checking the following:

- Ensure all cables are connected correctly
- Make sure the rocket launcher hasn't turned off
- Swap the IR LED out for a normal LED. It should flash if the irsend command is working
- Double-check the IR LED isn't plugged in backwards.

The Code FULL CODE LISTING

```
import pygame
import time
import RPi.GPIO as GPIO
import subprocess

PS3_AXIS_LEFT_H = 0
PS3_AXIS_LEFT_V = 1
PS3_AXIS_RIGHT_H = 2
PS3_AXIS_RIGHT_V = 3

PS3_CROSS = 14

pygame.init()

j = pygame.joystick.Joystick(0)
j.init()

print 'Initialized Joystick : %s' % j.get_name()

DRIVEA0 = 17
DRIVEA1 = 24
STANDBY = 18
DRIVEB0 = 21
DRIVEB1 = 22
A0 = False
A1 = False
B0 = False
B1 = False
```

"LIRC supports multiple ways of controlling IR, so it needs to be configured to use the Raspberry Pi GPIO pins"

The Code FULL CODE LISTING

```
GPIO.setmode(GPIO.BCM)
GPIO.setup(DRIVEA0, GPIO.OUT)
GPIO.setup(DRIVEA1, GPIO.OUT)
GPIO.setup(STANDBY, GPIO.OUT)
GPIO.setup(DRIVEB0, GPIO.OUT)
GPIO.setup(DRIVEB1, GPIO.OUT)

GPIO.output(DRIVEA0, A0)
GPIO.output(DRIVEA1, A1)
GPIO.output(STANDBY, False)
GPIO.output(DRIVEB0, B0)
GPIO.output(DRIVEB1, B1)

threshold = 0.60

LeftTrack = 0
RightTrack = 0

def setmotors():
    GPIO.output(DRIVEA0, A0)
    GPIO.output(DRIVEA1, A1)
    GPIO.output(STANDBY, True)
    GPIO.output(DRIVEB0, B0)
    GPIO.output(DRIVEB1, B1)

try:
  GPIO.output(STANDBY, True)

  while True:
    events = pygame.event.get()
```

The Code FULL CODE LISTING

```
    for event in events:
        UpdateMotors = 0

        if event.type == pygame.JOYAXISMOTION:
            if event.axis == PS3_AXIS_LEFT_V:
                LeftTrack = event.value
                UpdateMotors = 1
            elif event.axis == PS3_AXIS_RIGHT_V:
                RightTrack = event.value
                UpdateMotors = 1

        if UpdateMotors:
            if (RightTrack > threshold):
                A0 = False
                A1 = True
            elif (RightTrack < -threshold):
                A0 = True
                A1 = False
            else:
                A0 = False
                A1 = False
            if (LeftTrack > threshold):
                B0 = False
                B1 = True
            elif (LeftTrack < -threshold):
                B0 = True
                B1 = False
            else:
                B0 = False
                B1 = False
```

"To run the 'irsend' command from Python we will be using the 'subprocess' module, and we'll be using the 'X' button on the controller to trigger it"

The Code FULL CODE LISTING

```
            setmotors()
        elif event.type == pygame.JOYBUTTONDOWN:
        if event.button == PS3_CROSS:
            subprocess.call(["irsend","SEND_ONCE","Bigtrak","fire"])


except KeyboardInterrupt:
    GPIO.output(STANDBY, False)
    GPIO.cleanup()
    j.quit()
```

The Code LIRCD.CONF

```
begin remote

  name  Bigtrak
  flags RAW_CODES
  eps            30
  aeps          100

  ptrail       2026
  gap          60000

      begin raw_codes

          name fire
              3071     981    4053    2026    2026    4053
              4053    2005    2026    4053    2026   10154
              3050     981    4074    2005    2026    4053
              4053    2005    2026    4053    2026   10154
              3050     981    4053    2005    2026    4053
              4053    2026    2026    4053    2026   10133
              3050     981    4053    2005    2026    4053
              4053    2005    2026    4053    2026

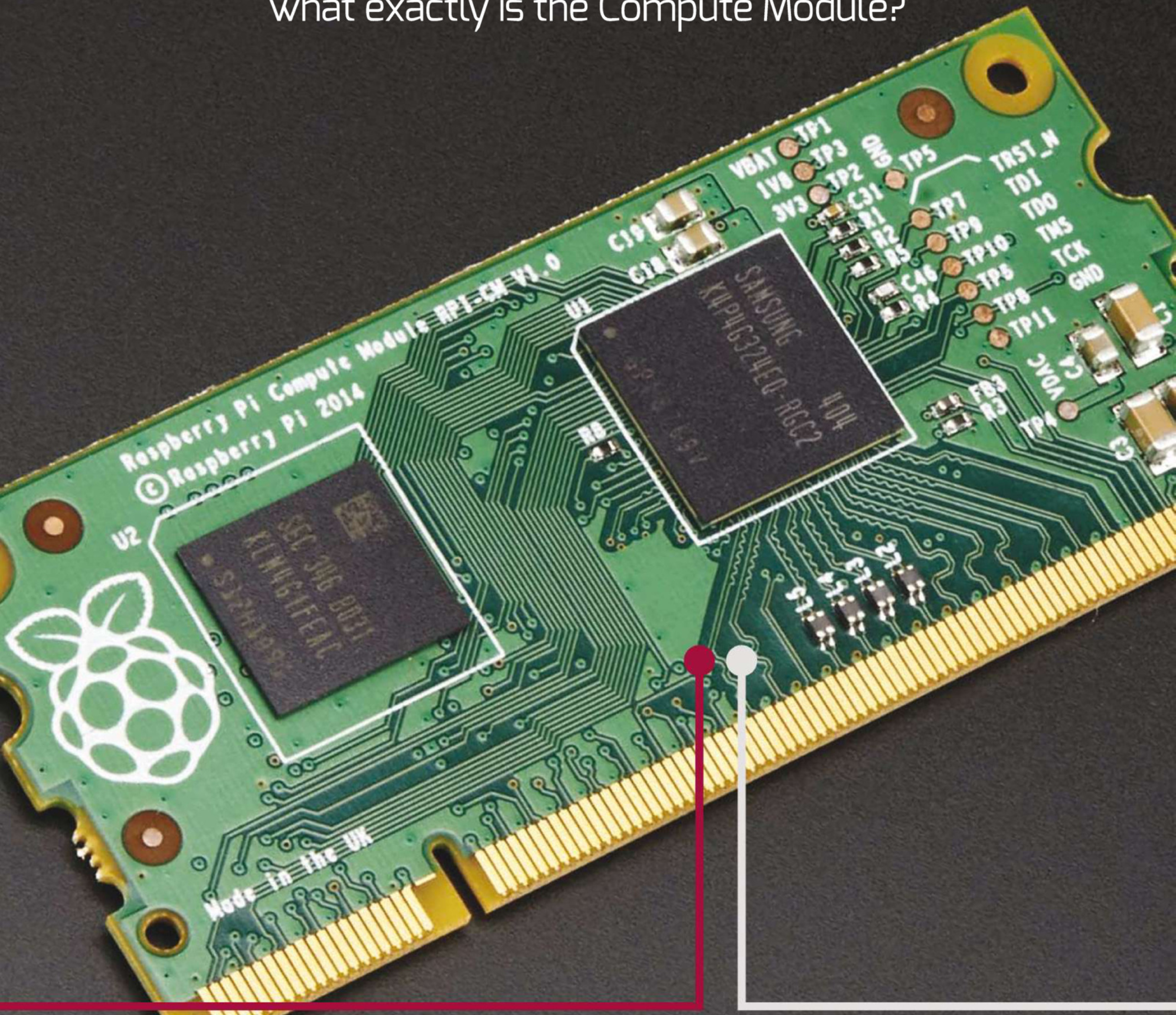
      end raw_codes

end remote
```

"Record the IR pulses sent by an unmodified Bigtrak when it triggers its 'fire' instruction. This will generate a configuration file that, after a few minor tweaks, looks like this"

What is the Raspberry Pi Compute Module?

You can now get the Raspberry Pi as a stick – but what exactly is the Compute Module?



Q I like the Raspberry Pi website's new redesign! It's very nice. I went there to have a look at it and they'd announced a new Raspberry Pi. Is it a new Raspberry Pi model? It looked like a stick.

A That would be the Raspberry Pi Compute Module. And no, the Compute Module version of the Raspberry Pi is not a new Raspberry Pi model or the oft-rumoured Raspberry Pi Model C.

Q Is it a Raspberry Pi 2 then?

A Also no, it's just a radically different form factor of the Raspberry Pi with the same chips and such that the original version had.

Q Wait, there's a 'rumoured' Model C Raspberry Pi? What would that be?

A Well there are currently two models of Raspberry Pi, ignoring the compute module. The Model A and Model B are basically the same, albeit with slightly different hardware configurations. The Model A only has one USB port and no Ethernet, whereas the Model B has two USB ports (the B+ has four), the Ethernet port and more RAM. The Model C is a predicted version of the Raspberry Pi that adds Wi-Fi. It's purely speculation though, as there's no real evidence anywhere to suggest this is happening.

Q Right, so it's not a new model. Why is it called the Compute Module then?

A It's literally just the computational chips that come on the Raspberry Pi along with some flash memory to replace the SD card. It can do the same work as a Raspberry Pi would because it has the same chips and can therefore run the same software.

Compute Module configurations

How you can use the Compute Module and the attached I/O board

The module

The Compute Module can be used on its own in custom hardware, slotting into a standard DDR2 SODIMM connector. With it, you have access to all the Raspberry Pi's computational functions along with on-board flash memory to install a distro or custom OS to.



The I/O board

On its own the I/O board is not very useful, requiring the Compute Module to properly power it. It features the slot for the Compute Module, a HDMI port, one USB A port, an extended GPIO port, extra display connectors and power. There's also a micro USB B port that can be used as a USB slave.

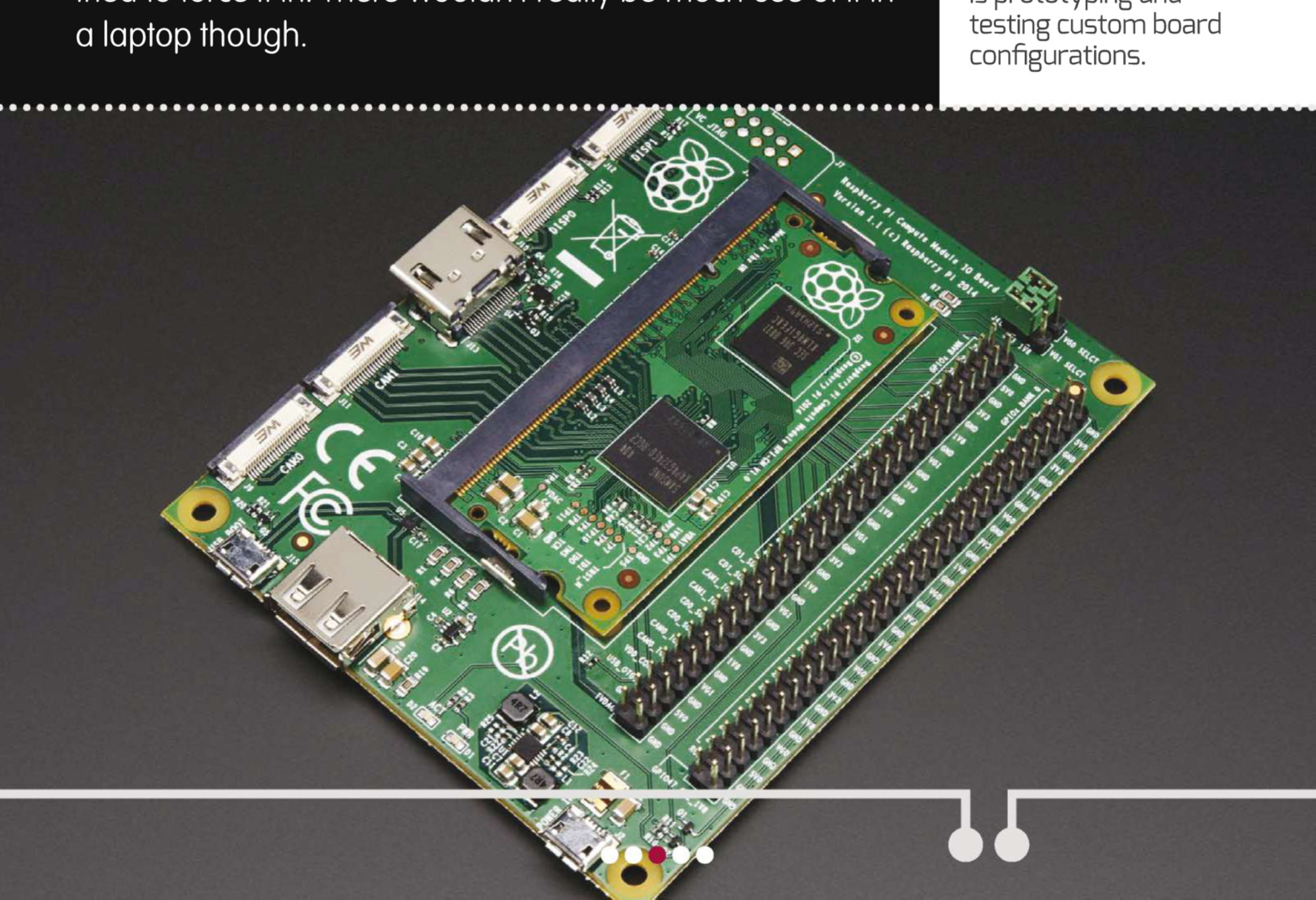
Q What does it offer over a normal Raspberry Pi, then?
A It's mainly the smaller form-factor, with the bonus of being a little bit cheaper than a full Raspberry Pi.

Q One thing I've noticed is that it doesn't seem to have any of the normal inputs and outputs. Neither does it have a power connector. How do you use it?
A The connectors on one side of the module are the same size as a DDR2 SODIMM, a standard that's used for laptop and mini-PC memory. You can create custom hardware that uses these standard parts to connect up these Raspberry Pi modules and save on space if needs be.

Q Laptop connectors? Does that mean I can plug it into my laptop?
A No no, it's not memory and it won't fit. You would probably break the module or the laptop or both if you tried to force it in. There wouldn't really be much use of it in a laptop though.

Compute Module configurations

Module-powered board (below)
With the I/O board and the compute module together you have a facsimile of a Raspberry Pi, working almost the exact same way as a normal Pi but with a different selection of inputs and outputs. The intended use for the I/O board combination is prototyping and testing custom board configurations.



Q So really there's no use for the Compute Module by a normal person at all?

A Not exactly. Along with the Compute Module there's also the Compute Module IO board, which you can plug the module into and get some of the inputs and outputs of the full Pi.

Q Some of the ports?

A The module IO board is a bit like the Model A board: one USB, no ethernet. However, it also comes with two extended GPIO ports and a micro USB slave port. It does lack the analog Video out and headphone jack though, along with the SD card slot that is a little redundant.

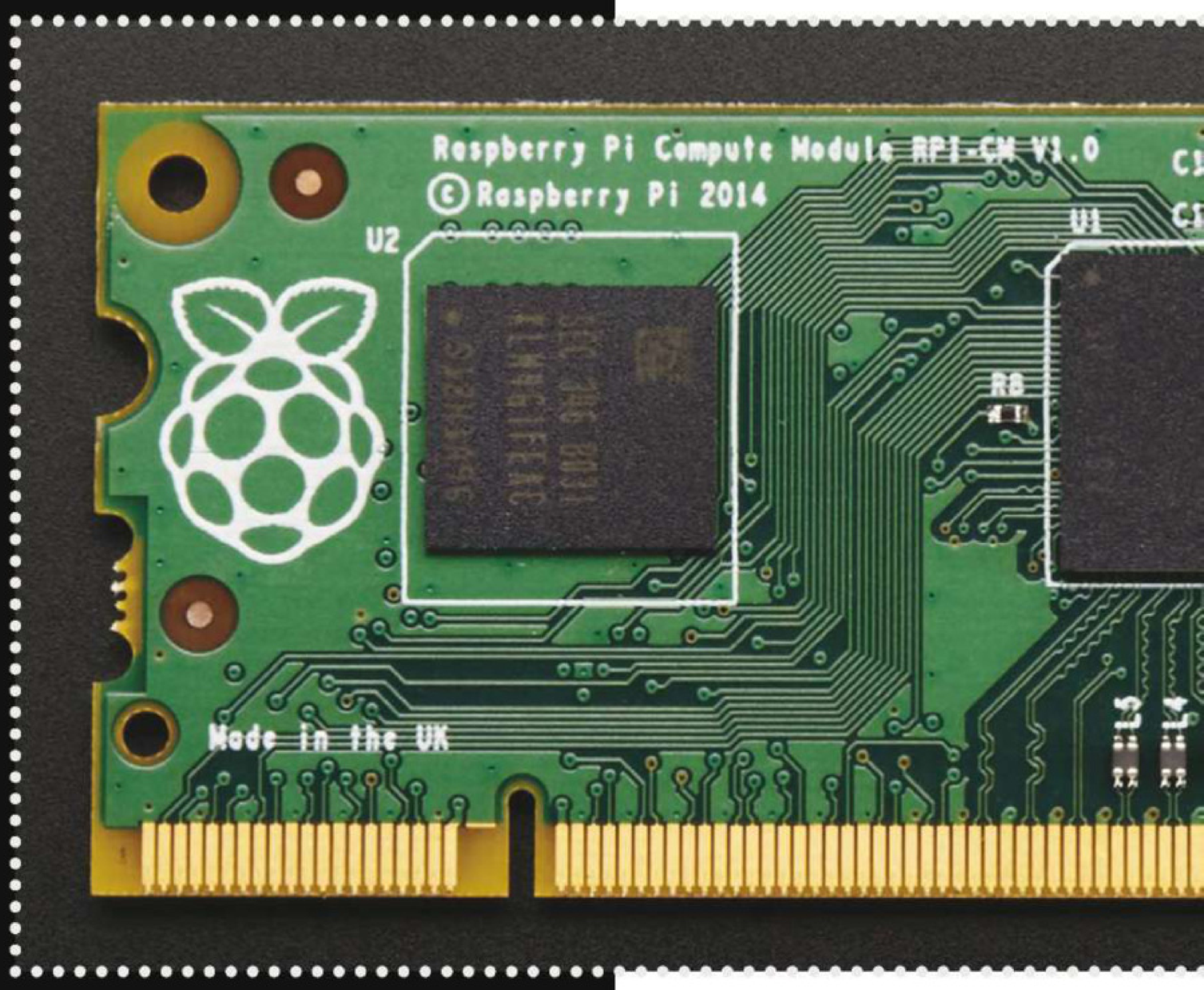
Q So could I just get these instead of a Raspberry Pi?

A You could but it would be more expensive than a normal Pi. You'd also miss out on the extra USB slot and ethernet port of the Model B, and it's also a little bit bigger than the normal Raspberry Pi.

Q What exactly would you use the I/O board for then?

A Well as we mentioned before, the Compute Module is mainly for people that would like to use the specific Pi processing parts in a smaller form-factor to build custom hardware. Before they can do that though, it helps to get prototyping so that developers can figure out exactly what they're connecting and how to do it. The I/O boards help with

Below If you build custom hardware, this could be for you



this by offering just about every useful input and output and the bare minimum to display the Raspberry Pi on a screen if needs be.

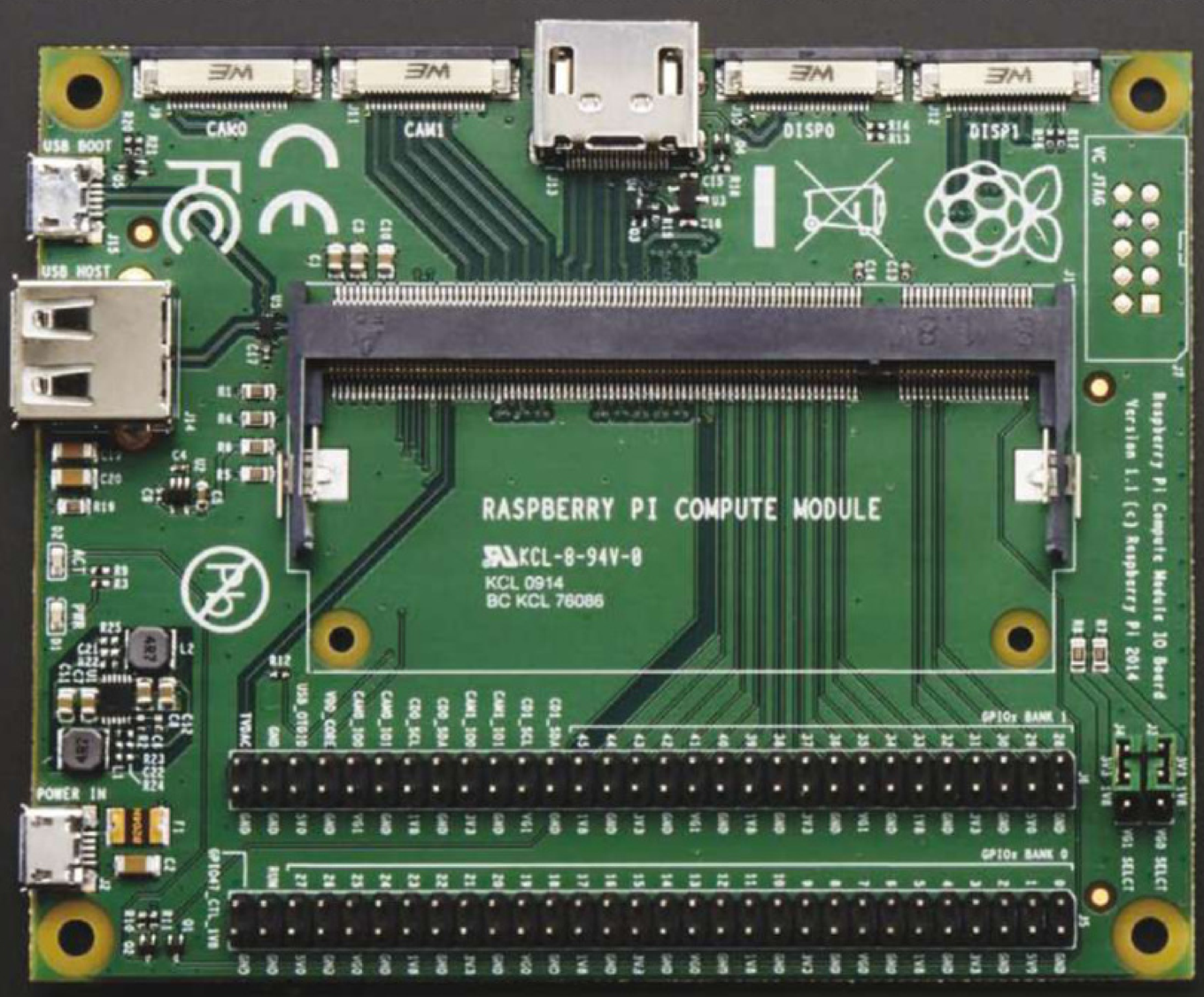
Q Ah, so it's a lot more for developers than for normal users. Is there anything stopping normal users from getting it?

A Not really, no. It may have potential for being able to do more project work thanks to extra GPIO pins and an extra camera input, so it definitely wouldn't be an awful purchase – just not as flexible or cheap as a Model B.

Q Fair enough. Say I'm a developer or a hobbyist with a need for the extra functions, where should I get one of these Compute Modules and boards?

A By the time you read this they should be available via the normal Raspberry Pi channels. So starting with RS Components and element14 before moving on to Pimoroni, ModMyPi and Adafruit. You can pick one up from element14 for around £28, while if you'd like to opt for a full compute module development kit then you can expect to pay £127.

> "It may have potential for being able to do more project work thanks to extra GPIO pins and an extra camera input"

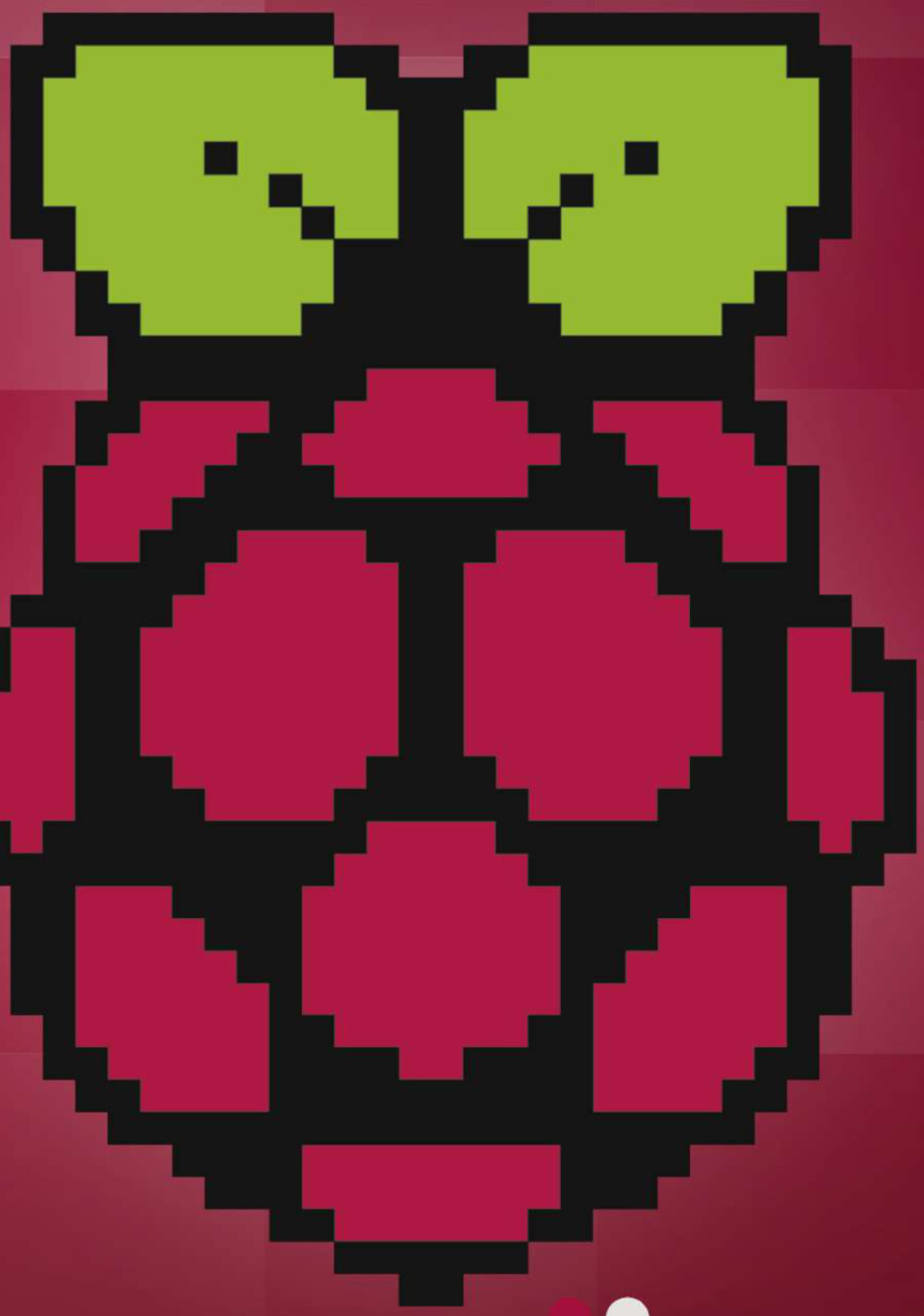




Below The full kit also contains a display and camera adapter, plus power, data and jumper cables

Get started with Minecraft Pi

Learn to program while playing one of the greatest games ever made!



Minecraft is a revolutionary sandbox game in which the protagonist, known only as Steve, is tasked with surviving and thriving in a huge, blocky 3D world considerably bigger than planet Earth. Each one-metre segment of the world is represented by a block. The block could be any one of a number of items and the deeper you dig into the earth the rarer the items might be. It's your challenge to dig, build and survive against an army of nocturnal monsters by harnessing the items you discover.

Minecraft Pi removes the survival element of the game, leaving you in a kind of Lego construction mode, where you have an infinite supply of many of the game's items and free reign to build with them. Minecraft Pi is also the only version of Minecraft (which is available on just about every gaming platform in circulation) where the player is actively encouraged to hack the world using a simple application programming interface (API).

In this project we're going to show you how to set up Minecraft Pi and configure it so you can interact with Minecraft in a way you've never done before. This small project is just the tip of the iceberg...

THE PROJECT ESSENTIALS

Latest Raspbian Image
raspberrypi.org/downloads

Minecraft-Pi tarball

Keyboard & mouse

Internet connection

01 Requirements

Minecraft Pi requires you to be running Raspbian on your Raspberry Pi, so if you're not already running that, take a trip to raspberrypi.org and get it setup. It also requires you have X Window loaded too. Assuming you're at the command prompt, you just need to type startx to reach the desktop.

Functional & fun coding

There's nothing too taxing about our code. We've created a couple of simple functions (starting with def) and used if, else and while to create the logic.

02 Installation

Make sure you're already in your home folder and download the Minecraft Pi package with the following commands in a terminal window:

```
cd ~
wget  https://s3.amazonaws.com/assets.
minecraft.net/pi/minecraft-pi-0.1.1.tar.gz
```

To use it we need to decompress it. Copy the following into the terminal window:

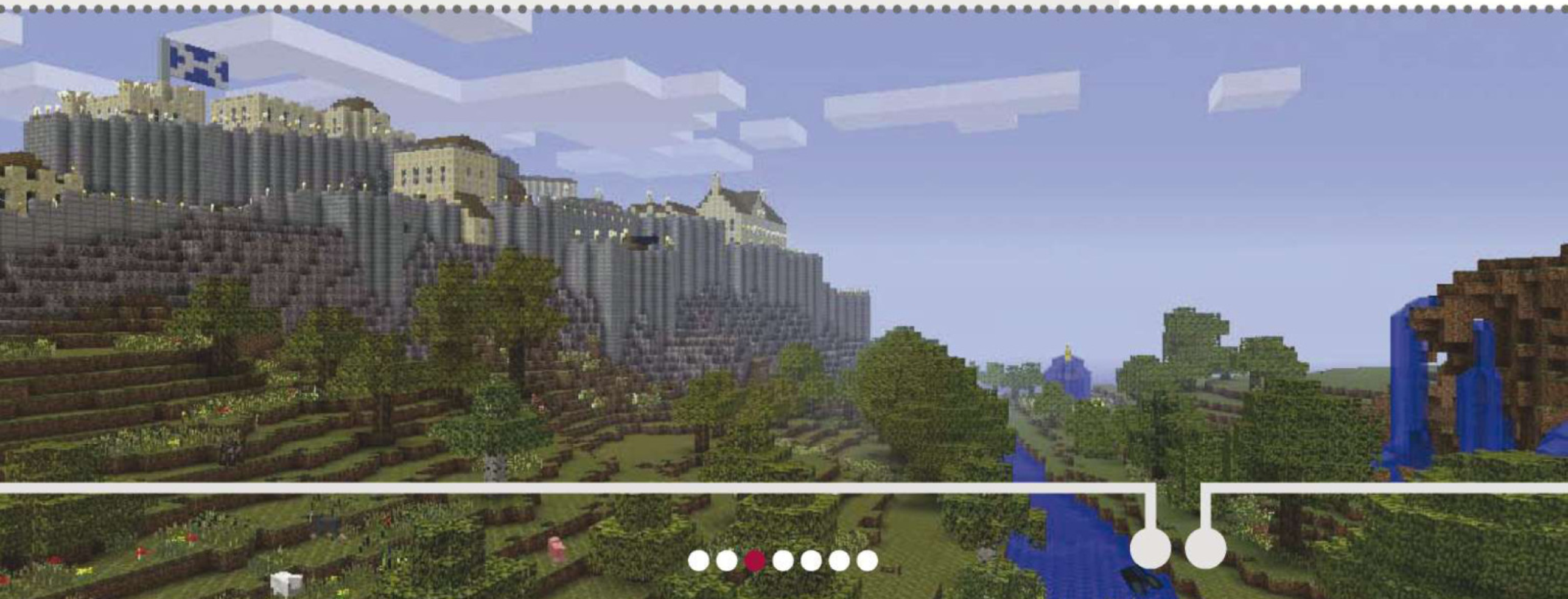
```
tar -zxvf minecraft-pi-0.1.1.tar.gz
```

Now you can move into the newly decompressed Minecraft Pi directory and try running the game for the first time:

```
cd mcpi
./minecraft-pi
```

03 Playing Minecraft Pi

Have a look around the game. If you're not familiar with Minecraft, you control movement with the mouse and the WASD keys. Numbers 1-8 select items in your quickbar, the space bar makes you jump and Shift makes you walk slowly (so you don't fall off edges). 'E' will open your inventory and double-tapping the space bar will also toggle your ability to fly.



Below Castles like this take an age to build... unless you have a script do the heavy lifting for you

04 Configuring the Python API

To take control of Minecraft with the Python API, you next need to copy the Python API folder from within the /mcpi folder to a new location. In the terminal, type the following:

```
cp -r ~/mcpi/api/python/mcpi ~/ minecraft
```

In this folder, we want to create a 'boilerplate' Python document that connects the API to the game. Write the following into the terminal:

```
cd ~/minecraft
nano minecraft.py
```

With nano open, copy the following and then save and exit with Ctrl+X, pressing Y (for yes), then Enter to return to the command prompt:

```
from mcpi.minecraft import Minecraft
from mcpi import block
from mcpi.vec3 import Vec3
mc = Minecraft.create()
mc.postToChat("Minecraft API Connected")
```

05 Testing your Python script

The short script you created contains everything you need to get started with hacking Minecraft Pi in the Python language. For it to work, you need to have the game already running (and be playing). To grab control of the mouse while in-game, you can press Tab. Open a fresh terminal window, navigate into your minecraft folder and start the script with the following commands:

"The short script you created contains everything you need to get started with hacking Minecraft Pi in the Python language"

Below Ever since September 2014, Minecraft Pi has been included in Raspbian



Download Pi Edition now!

Posted on December 20, 2012

MINECRAFT PI EDITION

The first iteration of Minecraft: Pi Edition is now available! And it's completely free to download. We're adding new features in due course, but thought you guys would appreciate us getting something out to you as soon as possible.

Daniel Frisk has provided the following instructions on how to get started.

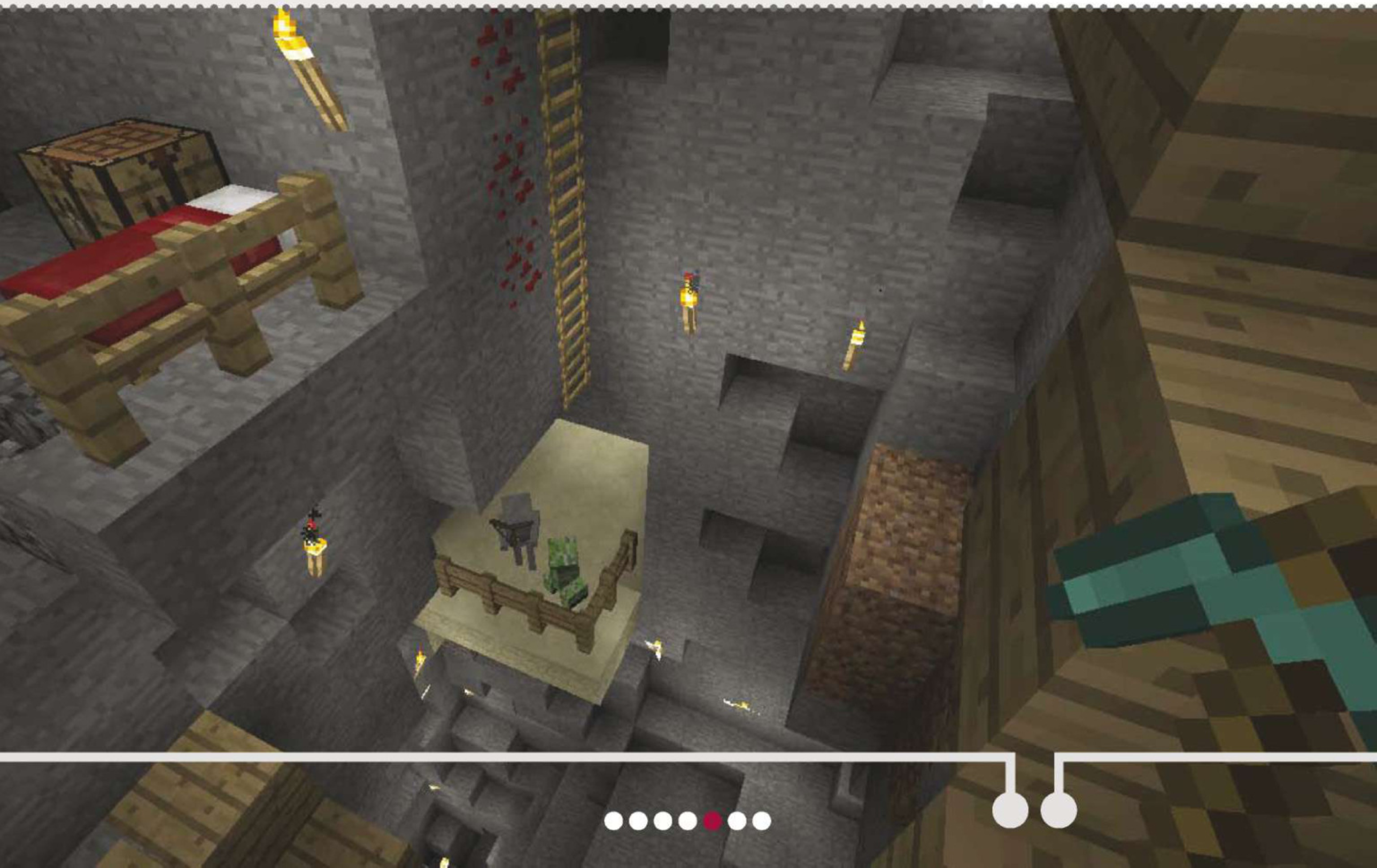
```
cd ~/minecraft
python minecraft.py
```

You'll see a message appear on screen to let you know the API connected properly. Now we know it works, let's get coding!

06 Hide & Seek

As you can see from the code starting on the next page, we've created a game of Hide & Seek adapted from Martin O'Hanlon's original creation (which you can find on **www.stuffaboutcode.com**). When you launch the script, you'll be challenged to find a hidden diamond in the fastest time possible. We've used it to demonstrate some of the more accessible methods available in the API, but there's much more to it than this demonstrates. Once you've got your head around it all, take a look at the guide to scripting that comes next.

Below Hide & Seek uses the warmer/colder system to let you know how far away you are



The Code FULL CODE LISTING

```
# !/usr/bin/env python

from mcpi.minecraft import Minecraft
from mcpi import block
from mcpi.vec3 import Vec3
from time import sleep, time
import random, math

mc = Minecraft.create() # make a connection to the game
playerPos = mc.player.getPos()

# function to round players float position to integer position
def roundVec3(vec3):
    return Vec3(int(vec3.x), int(vec3.y), int(vec3.z))

# function to quickly calc distance between points
def distanceBetweenPoints(point1, point2):
    xd = point2.x - point1.x
    yd = point2.y - point1.y
    zd = point2.z - point1.z
    return math.sqrt((xd*xd) + (yd*yd) + (zd*zd))

def random_block(): # create a block in a random position
    randomBlockPos = roundVec3(playerPos)
    randomBlockPos.x = random.randrange(randomBlockPos.x - 50, randomBlockPos.x + 50)
    randomBlockPos.y = random.randrange(randomBlockPos.y - 5, randomBlockPos.y + 5)
    randomBlockPos.z = random.randrange(randomBlockPos.z - 50, randomBlockPos.z + 50)
    return randomBlockPos
```

"When you launch the script, you'll be challenged to find a hidden diamond in the fastest time possible"

The Code FULL CODE LISTING

```
def main(): # the main loop of hide & seek
    global lastPlayerPos, playerPos
    seeking = True
    lastPlayerPos = playerPos

randomBlockPos = random_block()
    mc.setBlock(randomBlockPos, block.DIAMOND_BLOCK)
    mc.postToChat("A diamond has been hidden somewhere nearby!")
    lastDistanceFromBlock = distanceBetweenPoints(randomBlockPos, lastPlayerPos)
    timeStarted = time()
    while seeking:
        # Get players position
        playerPos = mc.player.getPos()
        # Has the player moved
        if lastPlayerPos != playerPos:
            distanceFromBlock = distanceBetweenPoints(randomBlockPos, playerPos)

            if distanceFromBlock < 2:
                #found it!
                seeking = False
            else:
                if distanceFromBlock < lastDistanceFromBlock:
                    mc.postToChat("Warmer " + str(int(distanceFromBlock)) + " blocks away")
                if distanceFromBlock > lastDistanceFromBlock:
                    mc.postToChat("Colder " + str(int(distanceFromBlock)) + " blocks away")
                lastDistanceFromBlock = distanceFromBlock
        sleep(2)
    timeTaken = time() - timeStarted
    mc.postToChat("Well done - " + str(int(timeTaken)) + " seconds to find the diamond")

if __name__ == "__main__":
    main()
```

Write scripts for Minecraft Pi

Get to grips with positioning and the co-ordinates system, block-making scripts and more

Minecraft is probably the biggest game on the planet right now. It's available on just about any format you can imagine, from PCs to gaming consoles to mobile phones. It should probably come as no surprise that it's also available on the Raspberry Pi. While at first glance Minecraft Pi is a simplified version of the Pocket Edition (designed for tablets and smartphones), the Raspberry Pi edition is very special, in that it's the only version of Minecraft to gives users access to its API (application programming interface). In this tutorial we're going to learn the basics...

THE PROJECT ESSENTIALS

Latest Raspbian Image
raspberrypi.org/downloads

Minecraft-Pi tarball

Keyboard & mouse

Internet connection

01 You must start X

There are a few things you need to bear in mind when coding and playing Minecraft on the Pi. Lots of people seem to get caught out by the fact the game won't work from the command line. Assuming that you haven't set your Pi to do so, you must start the desktop environment with startx. Launch the game from the terminal by entering the mcpi folder and typing:

```
./minecraft-pi
```

02 Desktop shortcut

Launching the game from the terminal isn't everyone's cup of tea. Coupled with the fact you need a separate terminal window open to launch your Minecraft Pi scripts (meaning you need two terminal windows open) it's worth creating a desktop shortcut to launch the game. Open a terminal and type:

```
cd ~/Desktop
```

Now create a new desktop file by typing: nano minecraft.desktop and copy the following into it:

```
[Desktop Entry]
Name=Minecraft Pi
Comment=Minecraft for the Pi
Exec=/home/pi/mcpi/minecraft-pi
Icon=/home/pi/mcpi/logo.png
Terminal=false
Type=Application
Categories=Games;Education;
StartupNotify=true
```

Press CTRL+X, Y and then press Enter in order to save the file and exit. The icon isn't in the mcpi folder by default, so you will need to do a Google Image search for the Minecraft icon, call it 'icon.png' and drop it into the folder. It needs to be 256 x 256 pixels and you may need to restart your desktop environment before the icon appears on your screen.

03 Tabbing out

One thing from our first Minecraft Pi tutorial (just before this one) that's worth reiterating is the fiddly nature of playing Minecraft Pi and starting scripts at the same time. Minecraft Pi needs to be running before you will be able to start a Python script that interacts with the game. This being the case, you will need to press Tab from Minecraft Pi in order to release the mouse so that you can then interact with the terminal window and load all of your Python scripts.

04 Read the docs!

You might think there is no documentation to help you code with Minecraft Pi, but you'd be wrong. Well, partially anyway. You see, one thing all good software scientists know is that the source code for a

"Coupled with the fact you need a separate terminal window open to launch your Minecraft Pi scripts, it's worth creating a desktop shortcut to launch the game"

program doubles up as a great instruction manual. For example, all the block types and names you can use in the game can be found by browsing through the block.py script found in the /mcpi/api/python/mcpi folder. Even better, you can find all of the functions you can carry out in the game simply by looking in 'minecraft.py' – they're even well commented so you know that setPos allows you to 'set entity position'.

05 A warning about scripts

What do you think you're going to call your very first Minecraft Pi script? There's an immediate temptation to call it 'minecraft.py' isn't there? Sadly, doing this can render the scripting aspect of Minecraft on the Raspberry Pi completely broken because you've potentially just overwritten the most important source code file in the game. Take a look in the /mcpi/api/python/mcpi folder – each of these file names is out of bounds!

Below Try naming scripts descriptively – like 'field.py' for your crop-creating code



06 Copy the API folder

The best way to keep the original API files of Minecraft Pi safe is to create a copy of the API folder that can be used separately. Do this by creating a new folder in your home folder and entering it (eg mkdir mc_projects && cd mc_projects) and then typing the following:

```
cp -r ~/mcpi/api/python/* .
```

The space followed by the full stop is required at the end of the command.

07 Your first script

Next we'll create a quick test script and make sure that we can successfully use the Python Minecraft API from its new location. Using your favourite text editor, type the following and save it as 'boilerplate.py':

```
from mcpi.minecraft import Minecraft
from mcpi import block
mc = Minecraft.create()
mc.postToChat("Minecraft API Connected")
```

If Minecraft Pi isn't running, start it now. When you're in the game, press Tab to release the mouse, open a terminal window and navigate to your Minecraft project folder.

Once inside, type python boilerplate.py and hit return – you will see the message 'Minecraft API Connected' appear as text chat within Minecraft Pi. We recommend that you start all of your Minecraft Pi scripts with this boilerplate code.

"When you're in the game, press Tab to release the mouse, open a terminal window and navigate to your Minecraft project folder"



Below Sadly, Minecraft Pi doesn't offer the excitement of underground caves and mobs, but it does allow you to generate dungeons by hacking

08 It didn't work?!

If at this point you're getting an error from Python, look back over your boilerplate code to ensure that everything's correct – nine times out of ten, it's because of a typo. If you're getting an import error you may have had an issue copying the API folder to the new location. Retrace your steps from the beginning and try again.

09 The fun starts here...

Now that we've successfully tested our boilerplate code we can start having real fun with the Minecraft Pi API and Python. There are a wide range of options for you to take advantage of here, such as getting and then setting blocks, finding and setting the player's position, changing and setting the camera and so on. The most commonly used API features, though, are the ones that revolve around manipulating the player's position and creating new blocks at set positions in the game world, to speed up construction.

10 The co-ordinates system

It's impossible to do anything in Minecraft Pi unless you're familiar with reading and interpreting the game's co-ordinate system. It's a fairly unusual system too, which switches the Y and Z co-ordinates in 3D space – in Minecraft Pi 'X' is sideways movement, 'Z' is forward movement and 'Y' is vertical movement. You can always find your current co-ordinates by glancing in the top-left corner of the screen, where they're displayed. It's worth taking a few minutes just to get familiar with the co-ordinates and how they change when you move around the game world.

Enter the Kano

If you've got a young Minecraft addict in your house that isn't quite ready to try out Python scripting, there is another avenue for you. Kano OS is a new operating system for the Raspberry Pi designed for younger children. Integrated into the OS is a modified version of Minecraft Pi, which uses a Scatch-like visual programming interface. It allows you to snap blocks together to manipulate the Minecraft world in much the same way as we have demonstrated here. You can learn more at **kano.me/downloads**

11 Origin point

When you load up a new Minecraft game for the first time, you'll start at the exact centre of the map, meaning your co-ordinates will be 0, 0, 0. Why is this useful? Because it makes building blocks and collections of blocks much easier for the beginner. At the origin point you don't need to work out where your surrounding blocks are – they're simply plus or minus 0.

12 getPos

Let's start out with one of the most simple operations with the API – finding your location. Knowing exactly where the player is in the game world is a surprisingly powerful and useful ability in Minecraft Pi. You can use it to help you do any number of things, including setting blocks near (or under) the player and triggering events (like finding specific locations). In the below example we'll find the player location and periodically set it to the variable my_pos and prove that it works by printing it to the chat window.

"Knowing exactly where the player is in the game world is a surprisingly powerful and useful ability in Minecraft Pi"

```
from mcpi.minecraft import Minecraft
from mcpi import block
import time

mc = Minecraft.create()
mc.postToChat("Minecraft API Connected")
while True:
    my_pos = mc.player.getPos()
    mc.postToChat("My position is:" + str(my_pos))
    time.sleep(1)
```

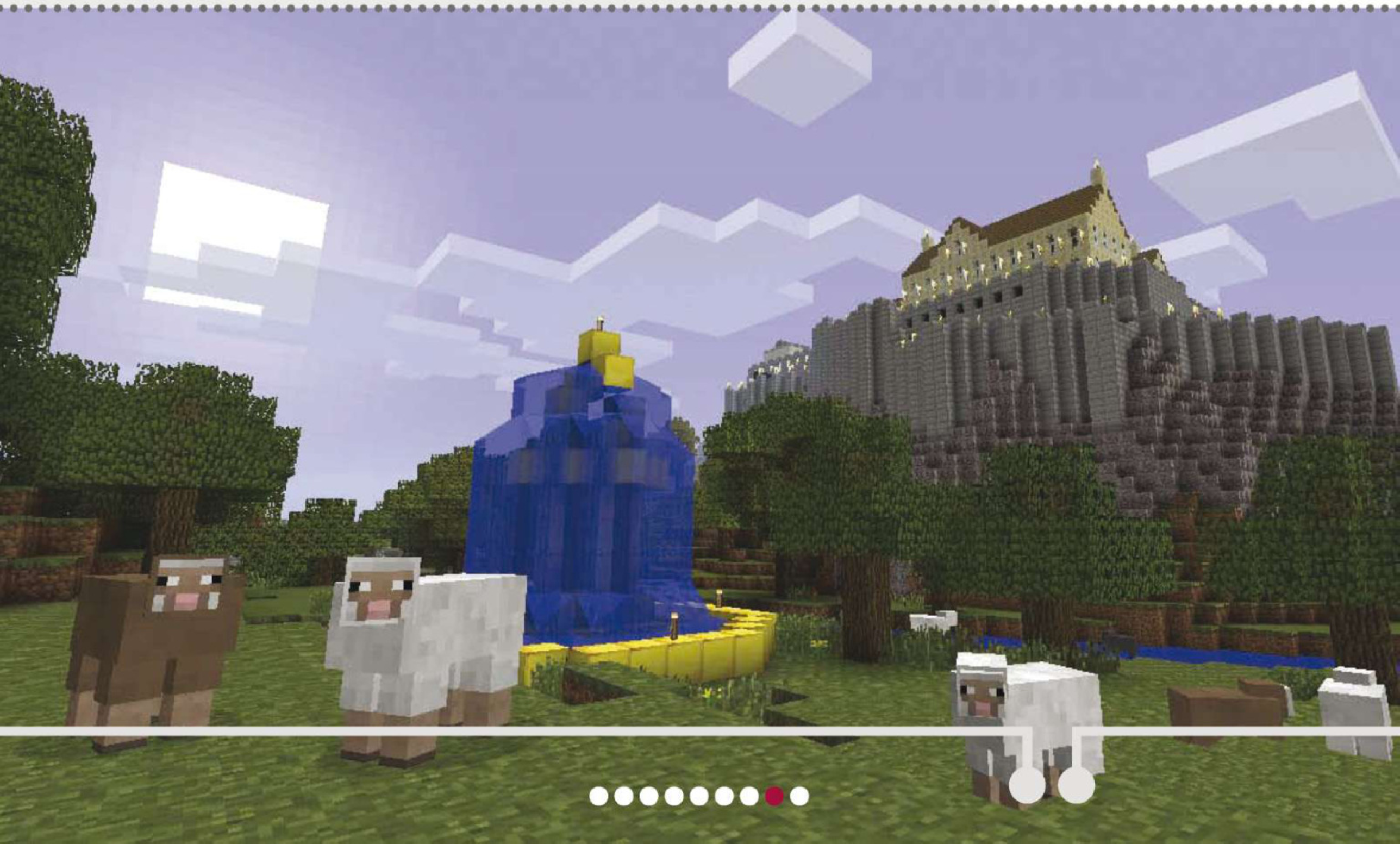
13 setBlock

The next most common action is setting a block to a certain type. You can find the full list of available blocks in the mcpi folder in a script called 'blocks.py'. We can append to our script in Step 12 by dropping a particular block type in our wake as we walk around the game world. After the 'mc.postToChat' line in our previous script add: mc.setBlock((my_pos), block.STONE) so the while loop section of the script now looks like this:

```
mc.postToChat("Stone path mode!")
while True:
    my_pos = mc.player.getPos()
    mc.setBlock((my_pos), block.STONE)
    time.sleep(1)
```

Now, when you activate this script when playing the game, you should notice that your character will drop

Below Still no sheep for Minecraft Pi, but fountains are no problem at all!



a block of stone every second at your current location. You should also consider making use of the setBlocks function to set a number of blocks simultaneously using the start co-ordinates and the end co-ordinates – this can help you on your way to creating much bigger structures.

> "Consider making use of the setBlocks function to set a number of blocks simultaneously using the start co-ordinates and the end co-ordinates"

14 Teleport Steve

So far we've read the location of the player and used it in the game, let's finish off by manipulating the player's location using the setPos method. In the following script we're going to draw the player back to the origin point, assuming they've wandered off and got lost:

```
from mcpi.minecraft import Minecraft
from mcpi import block
import time

mc = Minecraft.create()
mc.postToChat("Minecraft API Connected")

mc.player.setPos(0, 10, 0)
```

If you wanted to drop the player from a great height instead, you would change the Y co-ordinate to a large positive number. For example:

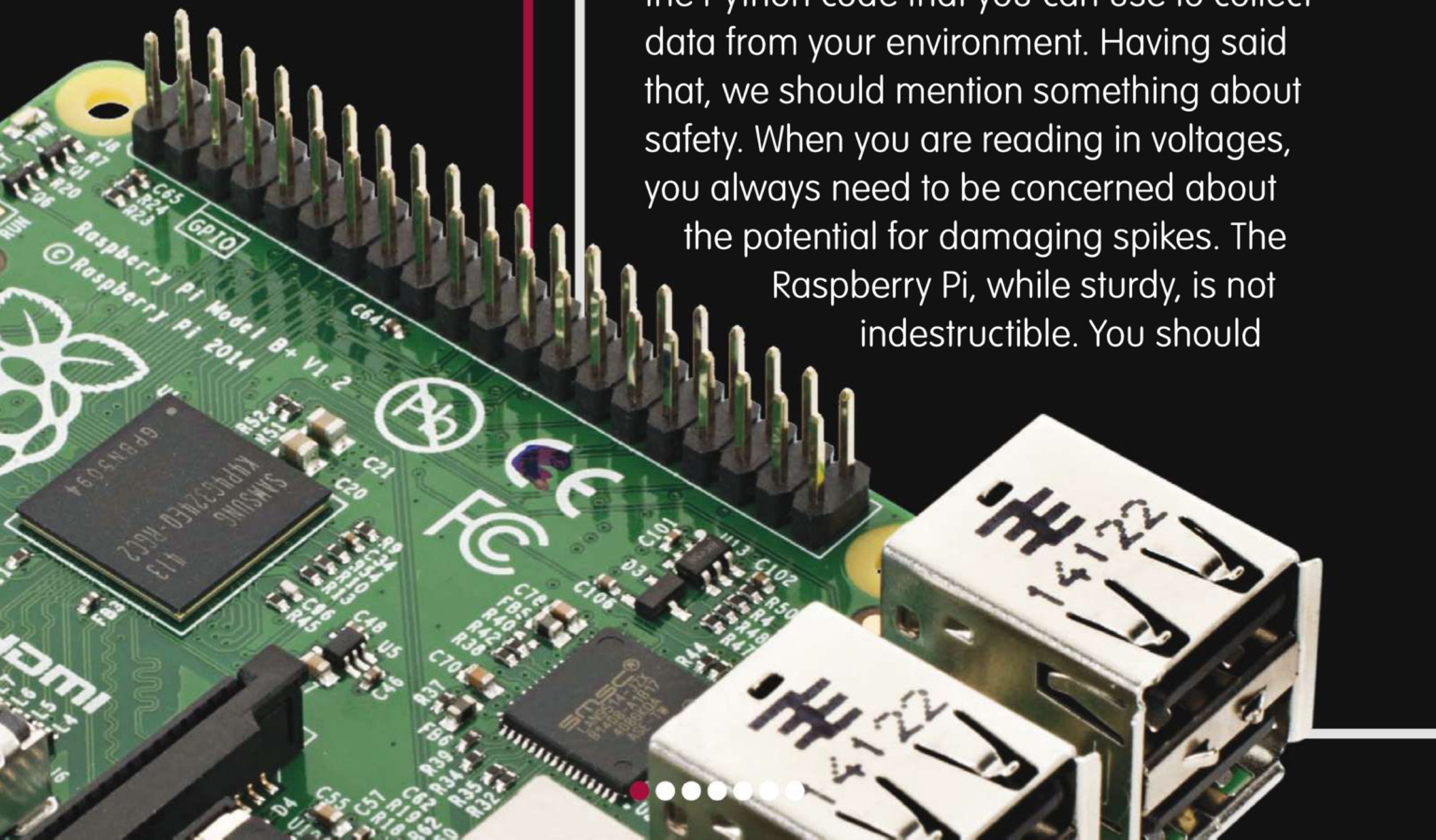
```
mc.player.setPos(0, 65, 0).
```

Looking at the world a different way

Discover how to read from the GPIO pins, ready to interact with the world using Python and a Raspberry Pi

"Always have a buffer between the outside environment and your Pi that will limit the maximum possible currents and voltages"

Last time, we looked at using the GPIO pins to send signals out from your Raspberry Pi. This month, we will look at using the input functions of these input/output pins. As with the last article, we won't be looking at the electronics side, but instead focusing on the Python code that you can use to collect data from your environment. Having said that, we should mention something about safety. When you are reading in voltages, you always need to be concerned about the potential for damaging spikes. The Raspberry Pi, while sturdy, is not indestructible. You should



always have a buffer between the outside environment and your Raspberry Pi that will limit the maximum possible currents and voltages. There are lots of examples online that offer some ideas.

Once you have your sensor wired up, it is time to start up your code and actually begin using the data that you are reading in. If you remember from issue four, the first step is to import the RPi.GPIO module. Remember that you can import this module as another shorter name to make the rest of your code easier to write and read. Then you will need to set the mode for the module, which will set the number scheme you will be using. The two options are either GPIO.BOARD or GPIO.BCM. The last step to set yourself up is to pick a pin and set up the channel.

At this point, we need to discuss pull-up and pull-down resistors. If the input pins are left alone, then they will tend to float and pick up any stray voltages that may be in the environment. Being inside a modern house, this is usually some stray electromagnetism from the mains wires. What we want is to have the pins tied either to high or low as the default voltage. In many cases, this is done in hardware where you tie the pin down physically to either ground or to the max voltage. You need to have a resistance in the circuit to either the high or low voltage, hence the name pull-up or pull-down resistor. The Raspberry Pi has the ability to define these internally in software. When you set up the channel, you can define an extra option, called pull_up_down, to be either high or low. This way, you can deal with floating inputs without the extra hardware. To do this, you can use either...

```
GPIO.setup(channel, GPIO.IN, pull_up_down=GPIO.
PUD_UP)
```

"If the input pins are left alone, then they will tend to float and pick up any stray voltages that may be in the environment"

...or...

```
GPIO.setup(channel, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
```

...where channel is the pin number, based on the mode that you set.

So, now that you have everything set up, you probably want to start reading data in. A key thing to know is that none of the GPIO pins are designed to handle analog input. This means that you can't directly measure things like voltages or currents. What you need to do is use some form of analog-to-digital circuit to convert any analog signal into a measurable digital signal that your Raspberry Pi can use. A simple signal is when a button is pressed. It is easy to wire a circuit so that when a button is pressed, you get either a low or high signal. You can use this to test your code separate to whatever circuitry you have to actually measure some environmental signal. The GPIO module includes a function called input() that reads some signal from a channel tied to a particular pin. Since the GPIO pins are digital, you will get either a 1 or true, or a 0 or false. So you can make decisions on either the numerical value or the boolean value.

Assuming that you have everything wired up correctly, you have two different techniques available to you that can be used to read this data. The first is straightforward polling. Essentially, you try to read from your input pin continuously, with some sort of a sleep statement between each attempt. Assuming that you set the pin up to have a pull-down resistance, so that the default value is low, then you can poll the pin with a piece of code like...

"You can't directly measure things like voltages or currents. You need to use some form of analog-to-digital circuit to convert any analog signal into a measurable digital signal"

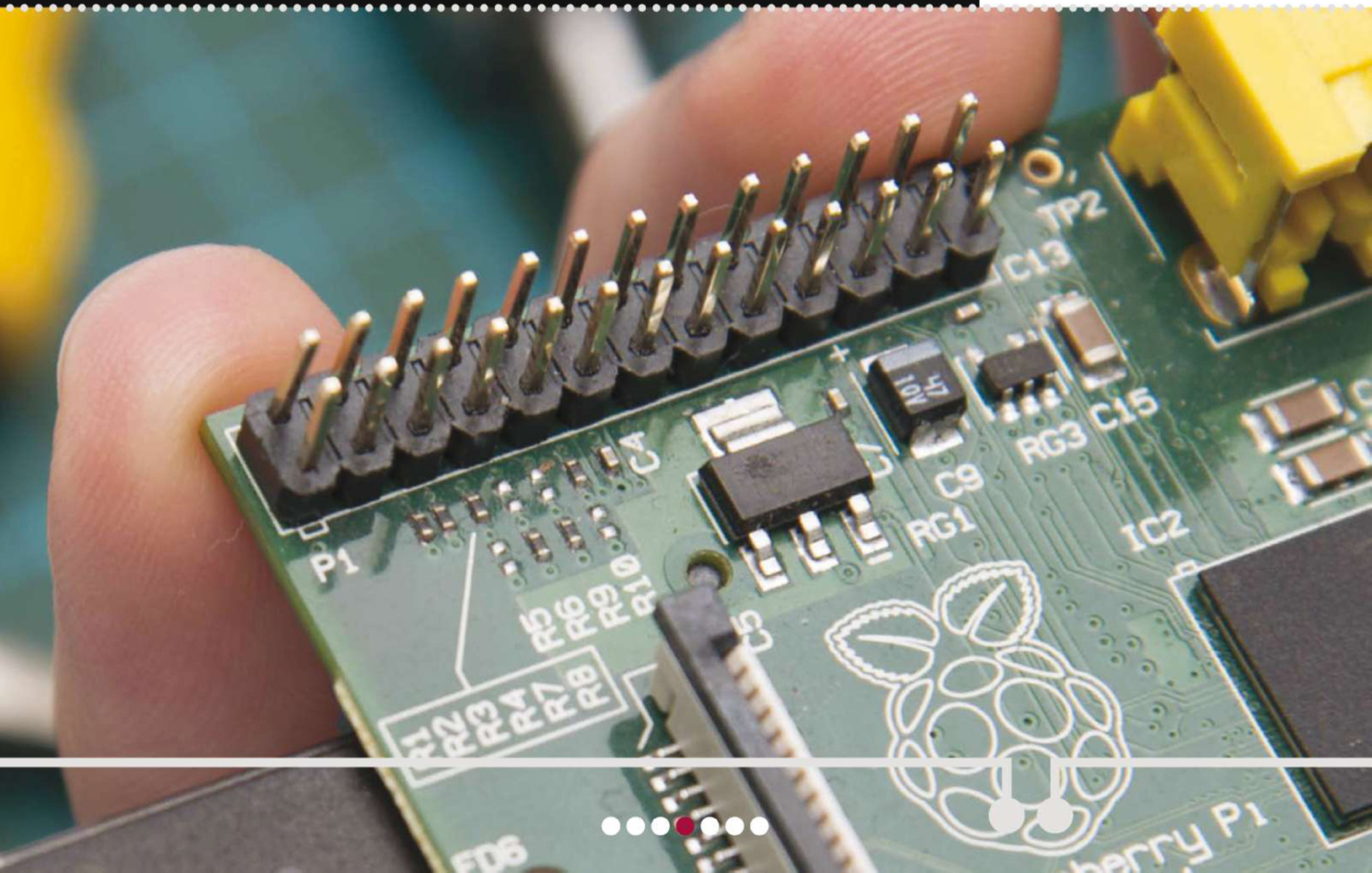
```
while GPIO.input(channel) == GPIO.LOW:
time.sleep(0.01)
```

In this case, the code checks to see whether the pin is low, and if so then it sleeps for 0.01s. While this code is easy to write, polling puts a fairly heavy load on the CPU. If you want to free up the CPU of your Raspberry Pi, you may want to consider using a callback function instead. Callbacks are handled in a single extra thread. This means that if you define more than one callback function, they will get executed sequentially. A basic callback function is defined with code like...

```
GPIO.add_event_callback(channel, my_callback_func)
```

...where my_callback_func is some function you have defined. This callback is triggered whenever a change is

Below None of the GPIO pins on the Raspberry Pi are designed to handle analog input



detected. If you want to only trigger your function either when the pin goes high or when it goes low, then you need to tie your callback function to one of these edges. For example, if you wanted to run your callback function whenever the pin goes high, you would use something like this:

```
GPIO.add_event_detect(channel, GPIO.RISING,
callback=my_callback)
```

You can equivalently trigger your callback function on GPIO.FALLING or GPIO.BOTH. When you have detected the event in question, you may want to stop the triggering of callbacks. You can do this with a call to GPIO.remove_event_detect(channel). Don't forget, once you have finished all of your interactions with the GPIO pins you need to clean up. You can do this by calling GPIO.cleanup(). If you only want to clean up the individual pins you were working with, you can do so with GPIO.cleanup(channel).

Now that you can both read and write from the GPIO pins, you are ready to use your Raspberry Pi to interact fully with the real world. Next time, we'll look at using matplotlib to visualise some of this interaction with the world at large. Until then, put some thought into how you can interact with the world at large and let your imagination run wild.

"If you want to only trigger your function either when the pin goes high or when it goes low, then you need to tie your callback function to one of these edges"

"Once you have finished all of your interactions with the GPIO pins you need to clean up using GPIO.cleanup()"

The Code FULL CODE LISTING

```
# The first step is to import the GPIO module
import RPi.GPIO as GPIO

# You need to pick a numbering scheme for the pins
GPIO.setmode(GPIO.BOARD)

# If you have hardware pull-up or pull-down
# resistors, you can set up a pin simply
GPIO.setup(12, GPIO.IN)

# If you want to use software, you can force
# input pins to default to high or low
GPIO.setup(13, GPIO.IN, pull_up_down=GPIO.PUD_UP)

# If you just want to read a single value in
GPIO.input(12)

# More likely, you need to poll until you get
# a particular value that you are looking for
while GPIO.input(13) == GPIO.LOW:
    time.sleep(0.01)

# If you want to instead wait for some change
# on a pin rather than a particular value
# you can detect the rising edge, lowering edge
# or both of a signal change
GPIO.wait_for_edge(12, GPIO.RISING)

# Instead of reading from a pin and blocking
# all other activity, you can define a callback
def my_callback(channel):
    print('This is my callback function')
```

The Code FULL CODE LISTING

```
GPIO.add_event_detect(13, GPIO.RISING)
GPIO.add_event_callback(13, my_callback)

# When you are done with the callbacks, you
# can remove it
GPIO.remove_event_detect(13)

# Clean up the pins you have used
GPIO.cleanup(12)
GPIO.cleanup(13)

# To use the I2C bus, you need import smbus
import smbus

# Create a new bus object
bus = smbus.SMBus(0)

# To send data to a device, you need an address
address = 0x50
bus.write_byte(address, 1)

# To read data or measurements, you can use
measurement1 = bus.read_byte(address)

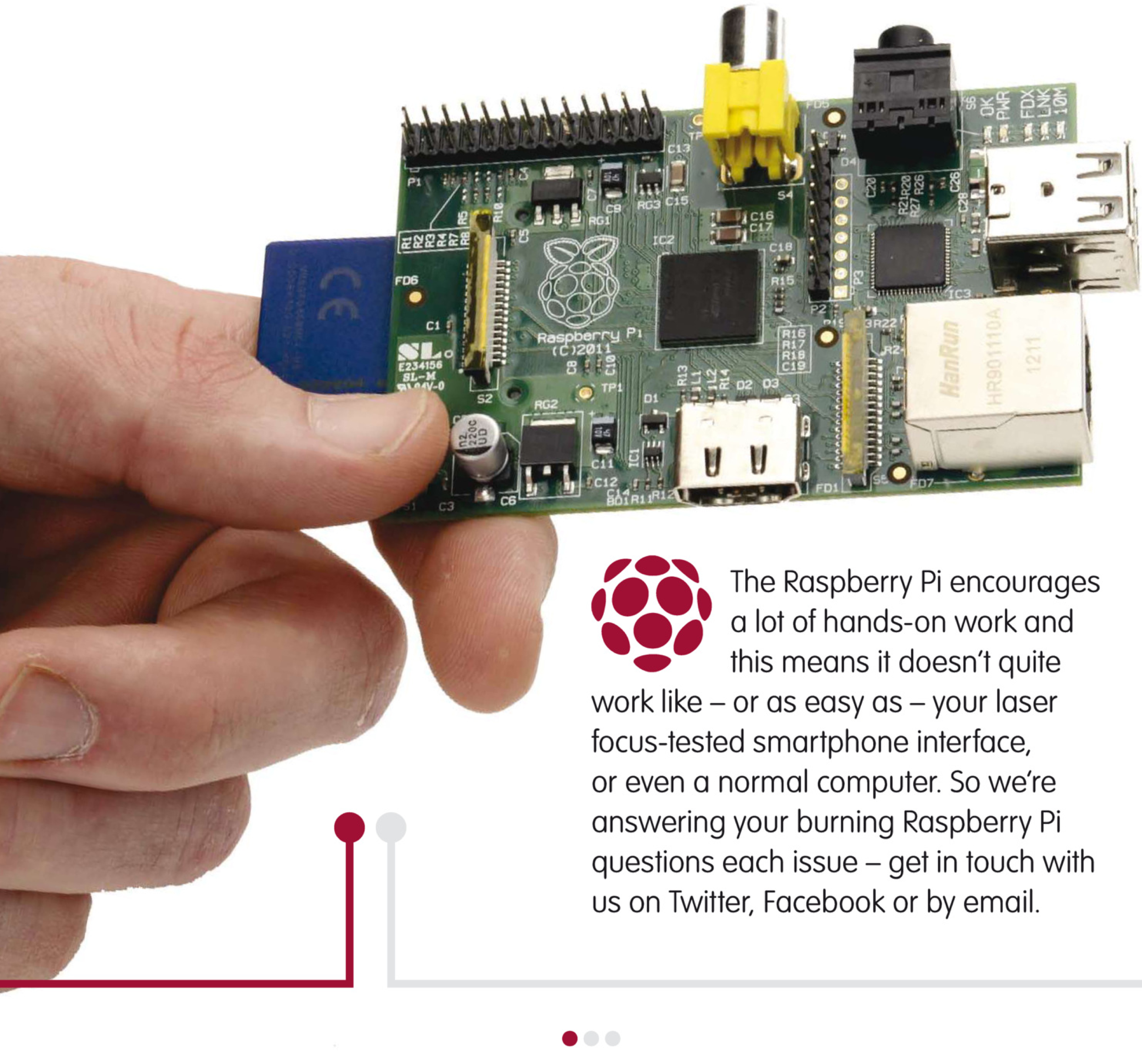
# When you are done, clean everything up
GPIO.cleanup()
```

> "If you only want to clean up the individual pins you were working with, you can do so with GPIO.cleanup(channel)"

Talking Pi

Join the conversation at...

@linuxusermag | Linux User & Developer | @ RasPi@imagine-publishing.co.uk



The Raspberry Pi encourages a lot of hands-on work and this means it doesn't quite work like – or as easy as – your laser focus-tested smartphone interface, or even a normal computer. So we're answering your burning Raspberry Pi questions each issue – get in touch with us on Twitter, Facebook or by email.

Is there any way to install Windows onto the Raspberry Pi?

Anna via Facebook

This is a bit of a tricky answer – technically you can, but you have to run it in a virtual computer (or a virtual machine as they're properly known) inside one of the Linux operating systems for Raspberry Pi. People have managed to get Windows 95 working on it, but it's been horrifically slow and basically unusable.

The quick answer is basically no, then. You can't install the latest version of Windows to the SD card, and whatever method you try will be basically unusable.

What is the audio quality like on the Raspberry Pi?

Chris via Twitter

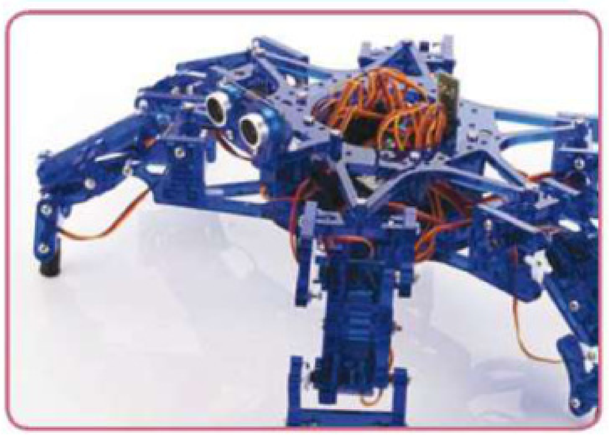
You can get audio three main ways from the Raspberry Pi. The standard 3.5mm headphone jack, a USB 'soundcard' and finally through the HDMI port. You can probably do it via the GPIO with a a bit of hackery but basically, quality-wise, it's best out of the HDMI. The 3.5mm jack gets better as you look at newer Pi's, but the HDMI port is designed for high definition video and audio, so as long as you've got high quality files and decent playback software, you can get great sounds from there.

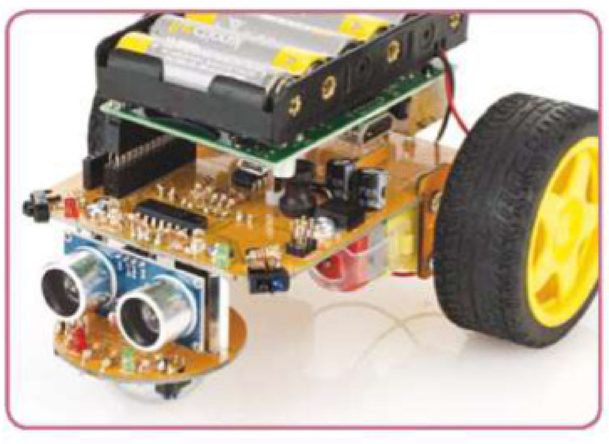
Follow @LinuxUserMag on Twitter. Search for the hashtag #RasPiMag

Win Ras Pi ROBOTS

Our sister mag, **Linux User & Developer**, just featured a massive guide to the very best robotics kits available right now that can be programmed with your Raspberry Pi – the whole thing was based on CamJam's recent Pi Wars event. You can download a copy now from iTunes (**bit.ly/1BLuU09**) or the Play Store (**bit.ly/1uJHsQt**).

The competition is wide open, though – enter all of the prize draws now! >>





Can you connect Raspberry Pis together to make them more powerful?
Prateek via email

There are a few Raspberry Pi 'supercomputer' methods that allow you to share load over the network, allowing for multiple Raspberry Pis to work together on specific tasks. We'll probably have one inside **RasPi** soon enough, but in the meantime give 'Raspberry Pi supercomputer' and 'Beowulf cluster' a Google and that should get you started. It's completely scalable as well, allowing you to use two Pis or lots.

If the Model B+ and A+ are now out, does that mean the Raspberry Pi 2 is on its way?
Toni via Facebook

We really have no idea – the B+ release was done with very little pre-release marketing or teasing or any kind of PR, and while it was sort of common knowledge that the A+ was coming there were no set release dates for it. For a Raspberry Pi 2, there may be some news about it before it comes out, but if they're going a Model C route with extra features then we may not know about it until it comes out.

Win Ras Pi ROBOTS

>> All you need to do is head to each robot's competition page and answer a quick question for a chance to win.

- Rover 5 **bit.ly/1ruvbVc**
- Pi2Go Lite **bit.ly/1z1p7m0**
- Hexy the Hexapod **bit.ly/1uNP7gd**
- Frindo **bit.ly/1z1pnBw**
- GoPiGo **bit.ly/1rAgbQ3**

More info:
See issue 147 of Linux User & Developer



FOR THE GNU GENERATION

www.linuxuser.co.uk



FREE DOWNLOAD EVERY ISSUE • FREE RESOURCE

LinuxUser & Developer

Available from all good newsagents & supermarkets today

ON SALE NOW:

» SUSECon '14 report » Raspberry Pi robots » Total online privacy



BUY YOUR ISSUE TODAY

Print edition available at www.imagineshop.co.uk

Digital edition available at www.greatdigitalmags.com

Available on the following platforms

Print | Tablet | Phone | Desktop

facebook.com/LinuxUserUK twitter.com/LinuxUserMag

Next issue

Get inspired · Expert advice · Easy-to-follow guides



Browse privately with Onion Pi

Plus Make a media centre, record slow-mo video and more

Get this issue's source code at:
www.linuxuser.co.uk/raspicode